

# Parallel I/O

**Katie Antypas**

**HPC Consultant**

**Lawrence Berkeley National Laboratory**

**ASTROSIM Summer School**

**July 19-23rd 2010**

Thanks to Rob Ross, Rob Latham from ANL and  
John Shalf for use of slides.





# Outline

## Day 1

- Review of storage systems and parallel file systems
- Parallel I/O interfaces
  - MPI-IO
  - Parallel I/O libraries
- Lustre Striping
- Best practices and recommendations

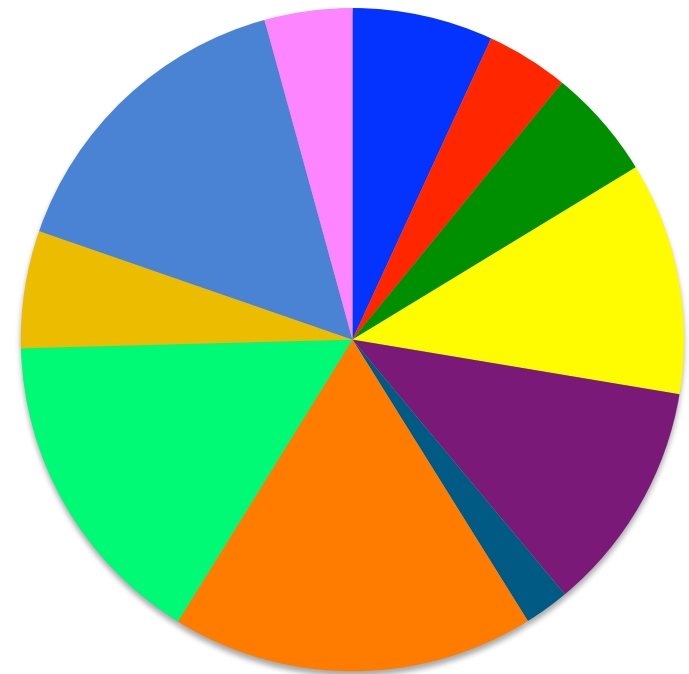
## Day 2 / Lab Session

- HDF5 parallel I/O library



# NERSC is the Primary Computing Center for DOE Office of Science

- **NERSC serves a large population**
  - Approximately 3000 users, 400 projects, 500 codes
  - Science-driven
  - Many collaborative international projects
- **Focus on “unique” resources**
  - Expert consulting and other services
  - High end computing systems
  - High end storage systems
- **NERSC is known for:**
  - Strong user services
  - Large and diverse user workload



■ Physics	■ Math + CS	■ Astrophysics
■ Chemistry	■ Climate	■ Combustion
■ Fusion	■ Lattice Gauge	■ Life Sciences
■ Materials	■ Other	



# NERSC Systems for Science

## Large-Scale Computing System

Franklin (NERSC-5): Cray XT4

- 9,532 compute nodes; 38,128 cores
- ~25 Tflop/s on applications; 356 Tflop/s peak

Hopper (NERSC-6): Cray XE6

- Phase 1: Cray XT5, 668 nodes, 5344 cores
- Phase 2: > 1 Pflop/s peak, (Summer 2010)



## Clusters

105 Tflops  
combined

Carver

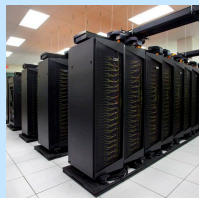
- IBM iDataplex cluster

PDSF (HEP/NP)

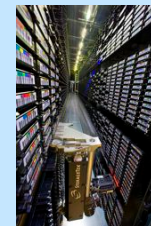
- Linux cluster (~1K cores)

Cloud testbed

- IBM iDataplex cluster



NERSC Global  
Filesystem (NGF)  
Uses IBM's GPFS  
1.5 PB; 5.5 GB/s



HPSS Archival Storage

- 40 PB capacity
- 4 Tape libraries
- 150 TB disk cache



## Analytics



- Euclid (512 GB shared memory)
- GPU testbed (48 nodes)



# Supernova Core-Collapse

**Objective:** First principles understanding of supernovae of all types, including radiation transport, spectrum formation, and nucleosynthesis.

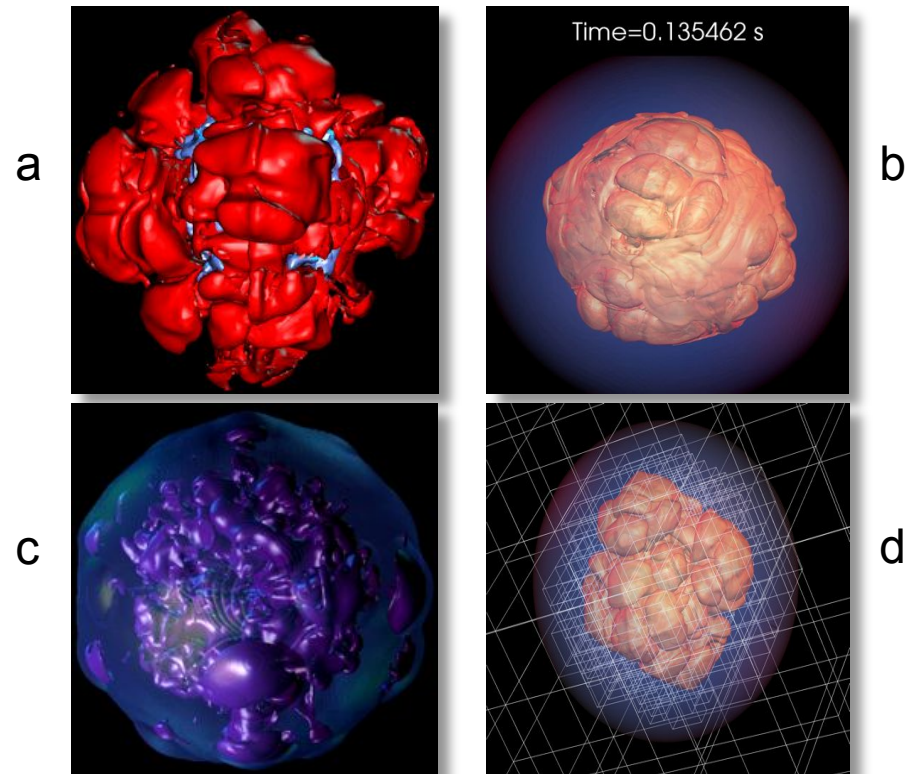
**Implications:** Will help us confront one of the greatest mysteries in high-energy physics and astrophysics -- the nature of dark energy.

**Accomplishments:** **VULCAN:** NERSC core collapse runs explain magnetically-driven explosions in rapidly-rotating cores.

- First 2.5-D, detailed-microphysics radiation-magnetohydrodynamic calculations; first time-dependent 2D rad-hydro supernova simulations with multi-group and multi-angle transport.

**NERSC:** 2M hours alloc in 2009; 2.2M used so far, requesting additional.

**PIs:** S. Woosley (UCSB),  
A. Burrows (Princeton)



*The exploding core of a massive star. a), b), and c) show morphology of selected isoentropy, isodensity contours during the blast; (d) AMR grid structure at coarser resolution levels."*





# High Energy Physics: Palomar Transient Factory

**Objective:** Process, analyze & make available data from Palomar Transient Sky survey (~300 GB / night) to expose rare and fleeting cosmic events.

**Implications:** First survey dedicated solely to finding transient events.

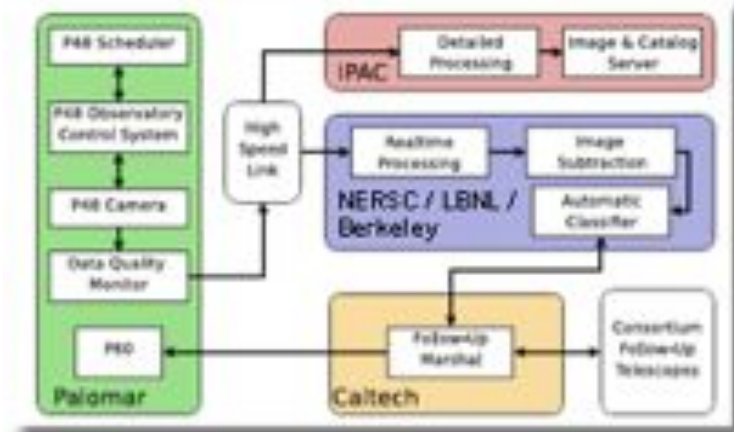
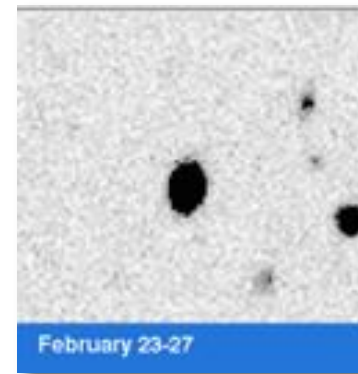
**Accomplishments:** Automated software for astrometric & photometric analysis and *real-time* classification of transients.

- Analysis at NERSC is fast enough to reveal transients *as data are collected*.
- Has *already uncovered* more than 40 supernovae explosions since Dec., 2008.
- Uncovering a new event about *every 12 minutes*.

## NERSC:

- 40k hours + 1M HPSS in 2009; Use of NERSC NGF + gateway

**PI: P. Nugent (LBNL)**



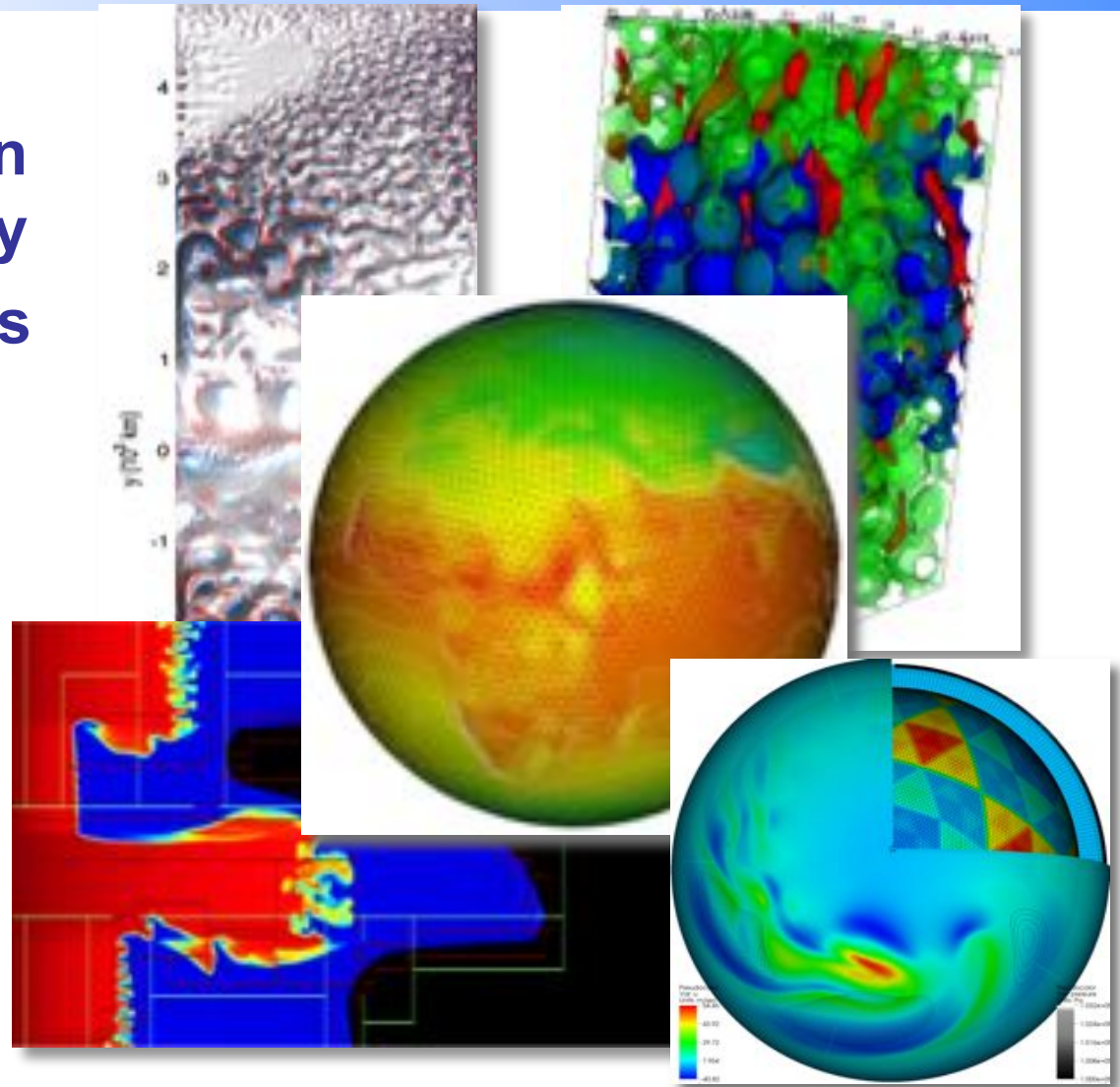
PTF project data flow

Two manuscripts submitted to Publications of the Astronomical Society of the Pacific



# Getting bigger all the time

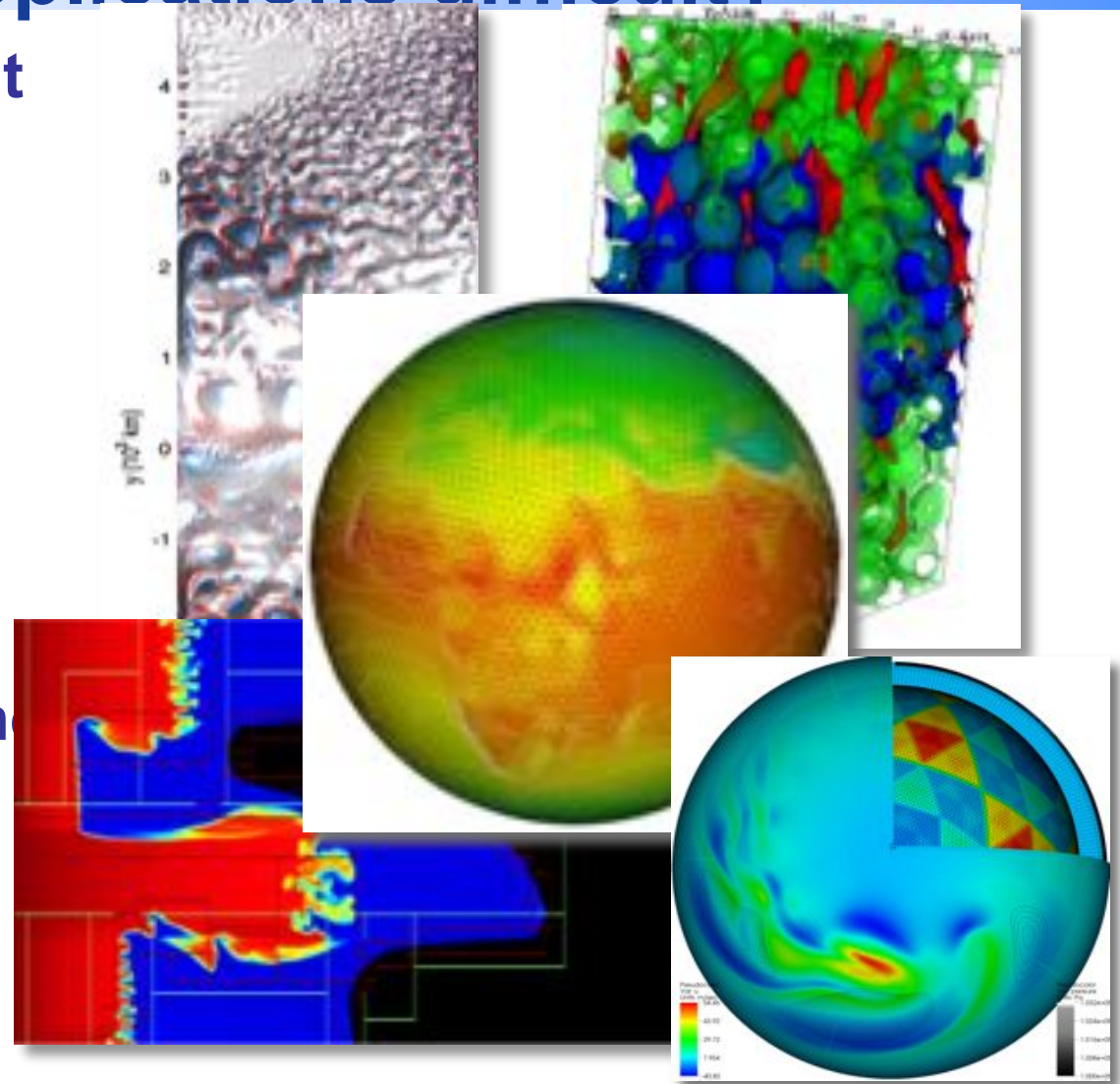
- User I/O needs growing each year in scientific community
- For our largest users I/O parallelism is mandatory
- I/O remains a bottleneck for many users





# Why is Parallel I/O for science applications difficult?

- Scientists think about data in terms of their science problem: molecules, atoms, grid cells, particles
- Ultimately, physical disks store bytes of data
- Layers in between, the application and physical disks are at various levels of sophistication







# Parallel I/O: *A User Perspective*

- **Wish List**
  - Write data from multiple processors into a single file
  - File can be read in the same manner regardless of the number of CPUs that read from or write to the file. (eg. want to see the logical data layout... not the physical layout)
  - Do so with the same performance as writing one-file-per-processor
  - And make all of the above portable from one machine to the next
- **Inconvenient Truth: Scientists need to understand about I/O in order to get good performance**



# I/O Hierarchy Slide

*Application*

*High Level IO Library*

*MPI-IO Layer*

*Parallel File System*

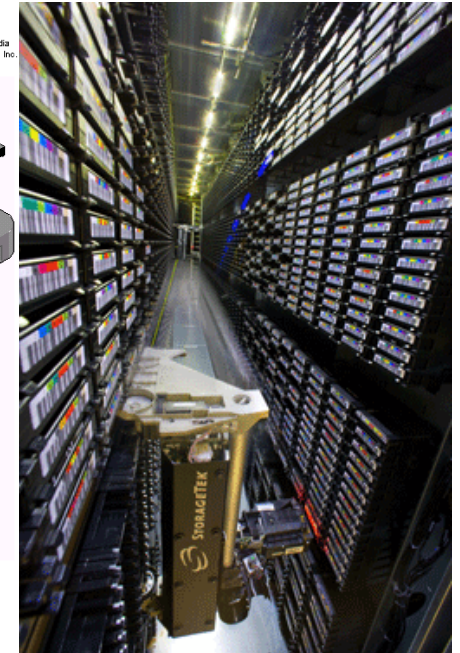
*Storage Device*



# Storage Media



*Magnetic Hard  
Disk Drives*



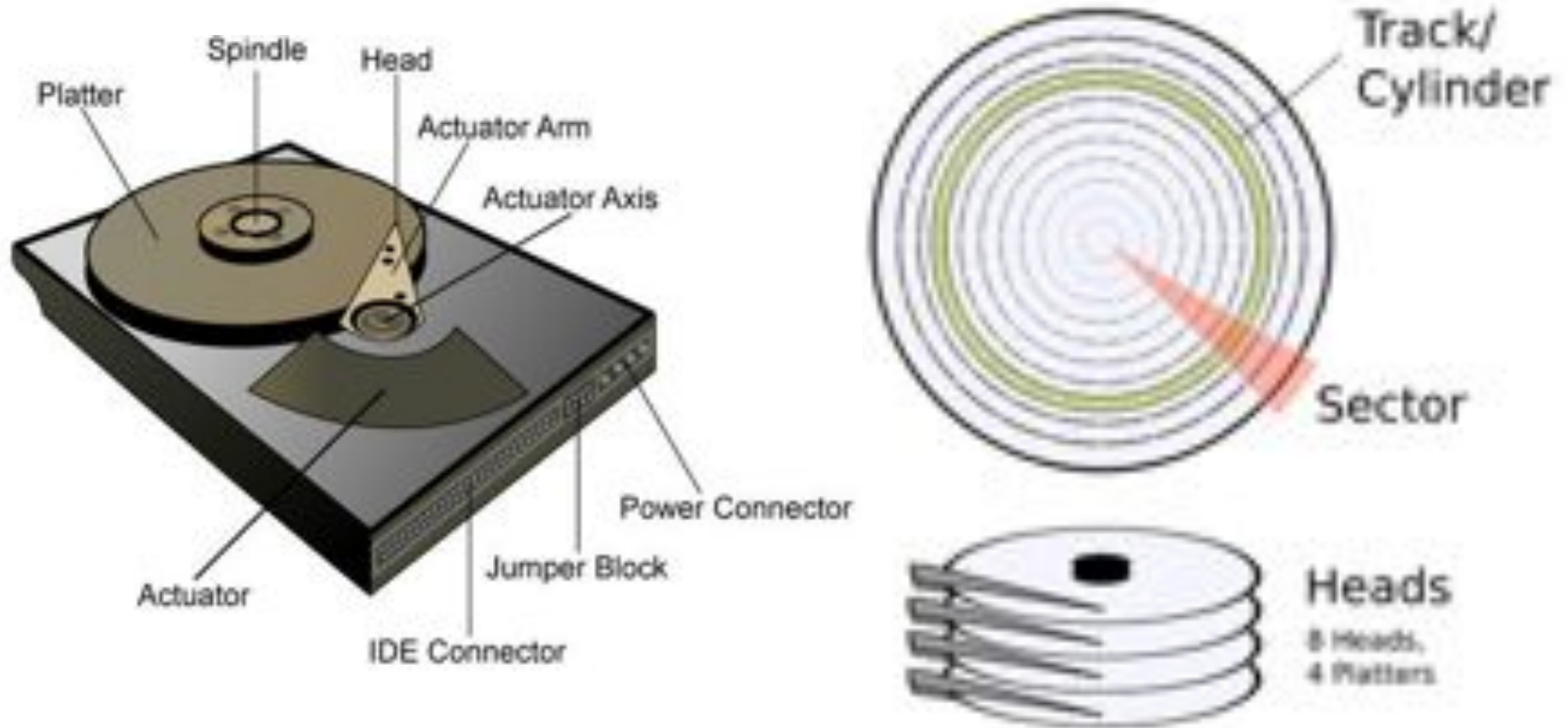
*Magnetic Tape*



*Solid State Storage Devices (flash)*



# Magnetic Hard Disk Drives

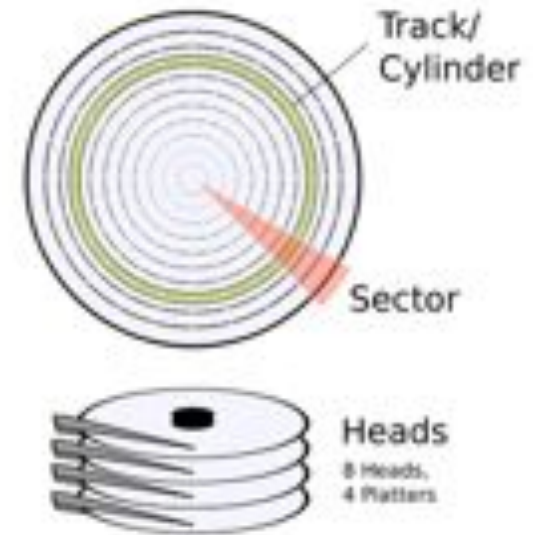






## Some definitions

- **Capacity (in MB, GB, TB)**
  - Measured by areal density
  - $\text{Areal density} = \text{track density} * \text{linear density}$
- **Transfer Rate (bandwidth) – MB/sec**
  - Rate at which a device reads or writes data
- **Access Time (milli-seconds)**
  - Delay before the first byte is read





# Access Time

- $T(\text{access}) = T(\text{seek}) + T(\text{latency})$ 
  - $T(\text{seek})$  = time to move head to correct track
  - $T(\text{latency})$  = time to rotate to correct sector

- $T(\text{seek}) = \sim 10 \text{ msec}$
- $T(\text{latency}) = 4.2 \text{ msec}$
- $T(\text{access}) = 14 \text{ msec!!}$
- How does this compare to clock speed?

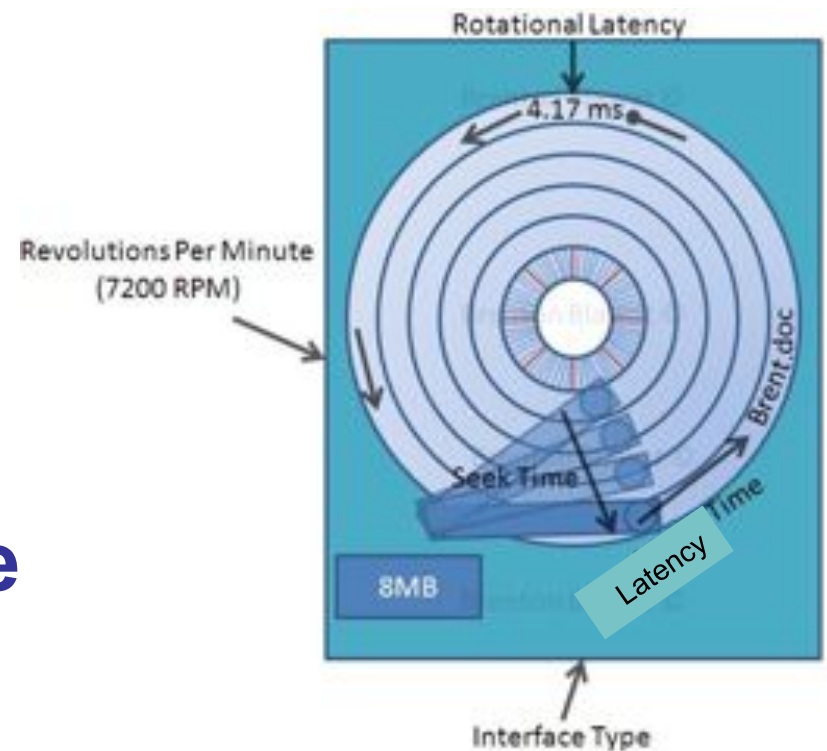


Image from Brenton Blawat



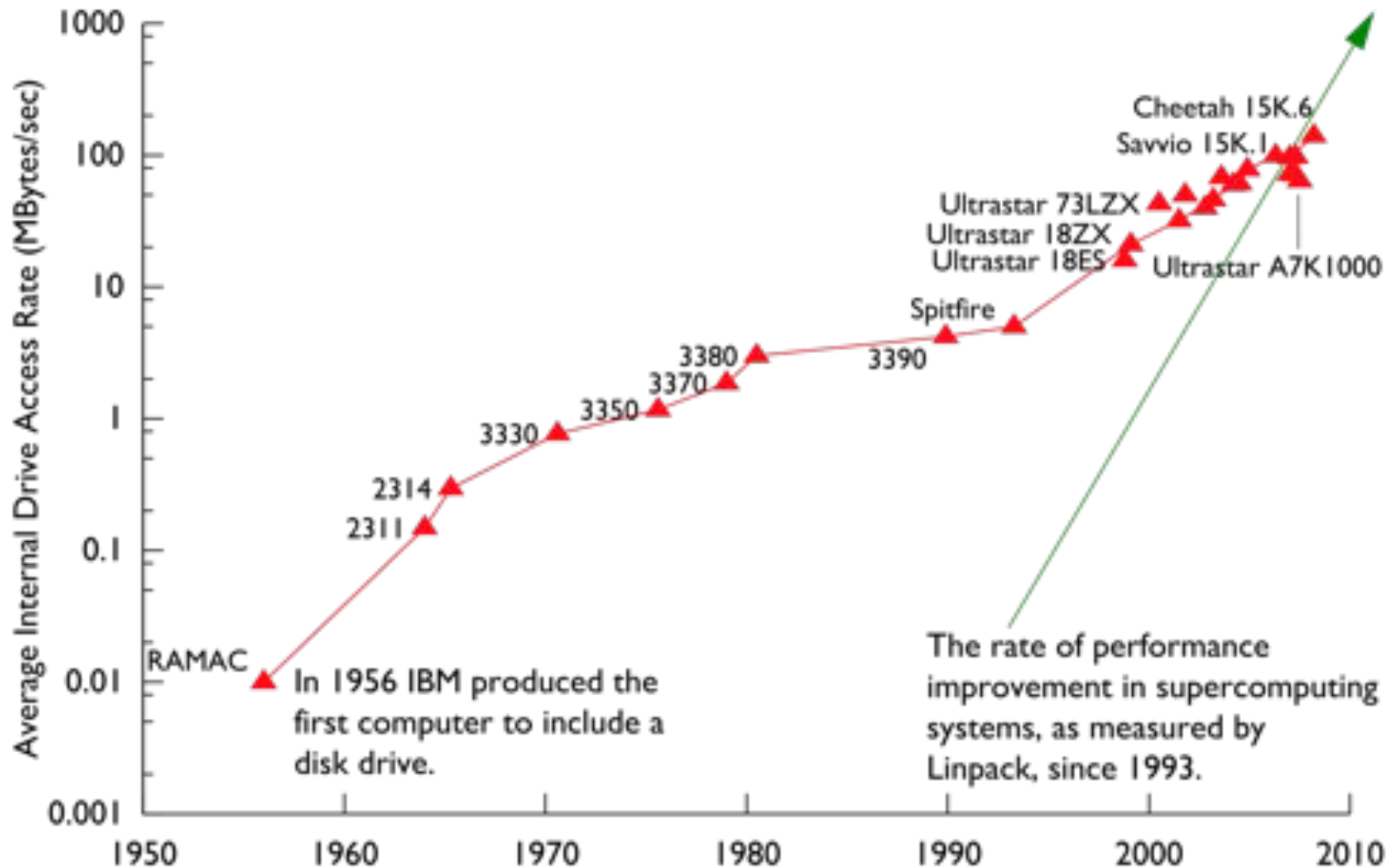
# Why isn't a hard drive faster?

- Hard disk drives operate at about 30-100s MB/sec
- Areal density getting higher
- Improved magnetic coating on platters
- More accurate control of head position on platter
- Rotational rates slowly increasing
- What's the problem?

- *Electronics supporting the head can't keep up with increases in density*
- *Access time always a fixed cost*



# Disk Transfer Rates over Time



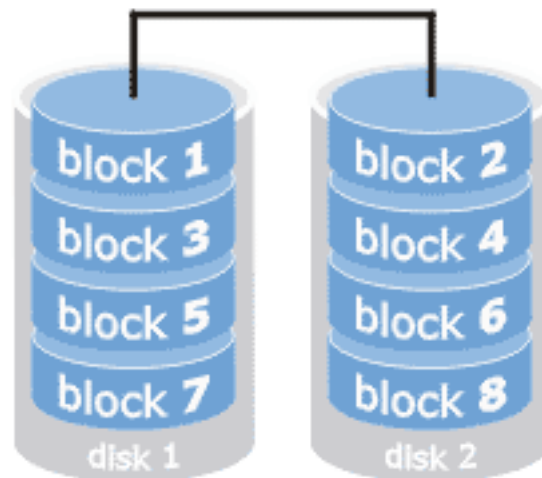
Thanks to R. Freitas of IBM Almaden Research Center for providing much of the data for this graph.





# RAID

- Individual disk drives obviously not fast enough for supercomputers
- Need parallelism
- Redundant Array of Independent Disks



- Different configurations of RAID offer various levels of data redundancy and fail over protection



# Solid State Storage

- **More reliable than hard disk drives**
  - No moving parts
  - Withstand shock and vibration
- **“Non-volatile” retain data without power**
- **Slower than DRAM memory, faster than hard disk drives**
- **Relatively cheap**





# File Systems



# What is a File System?

- **Software layer between the Operating System and Storage Device which creates abstractions for**
  - Files
  - Directories
  - Access permissions
  - File pointers
  - File descriptors
- **Moving data between memory and storage devices**
- **Coordinating concurrent access to files**
- **Managing the allocation and deletion of data blocks on the storage devices**
- **Data recovery**





# From disk sectors to files

- File systems allocate space in “blocks”, usually multiples of disk “sector”
- A file is created by the file system on one or more unused blocks
  - The file system must keep track of used and unused blocks
  - Attempts to allocate nearby blocks to the same file
- “inodes” are special blocks that contain a list of all the blocks in a file.



# Inodes store Metadata

- Data about data
- File systems store information about files externally to those files.
- Linux uses an inode, which stores information about files and directories (size in bytes, device id, user id, group id, mode, timestamps, link info, pointers to disk blocks, file size...)
- Any time a file's attributes change or info is desired (e.g., `ls -l`) metadata has to be retrieved from the metadata server
- Metadata operations are IO operations which require time and disk space.



# File Systems

- Your laptop or desktop has a file system, referred to as a “local file system”
- A networked file system allows multiple clients to access files
  - Treats concurrent access to the same file as a rare event
- A parallel file system builds on the concept of a networked file system
  - Efficiently manages hundreds to thousands of processors accessing the same file concurrently
  - Coordinates locking, caching, buffering and file pointer challenges
  - Scalable and high performing



# There are a number of parallel file systems

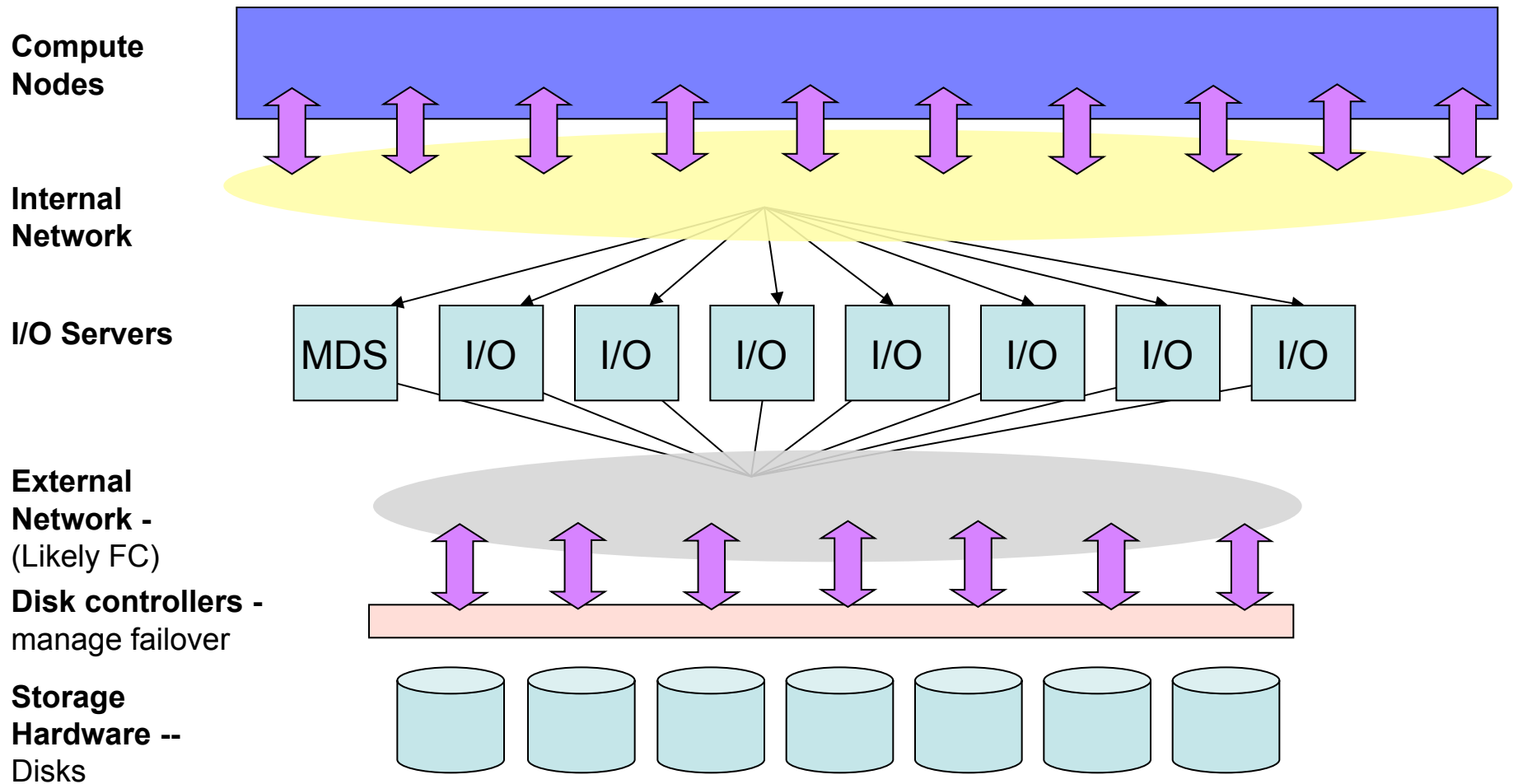
- These are three parallel file systems widely used on the top 500 supercomputers



# GPFS



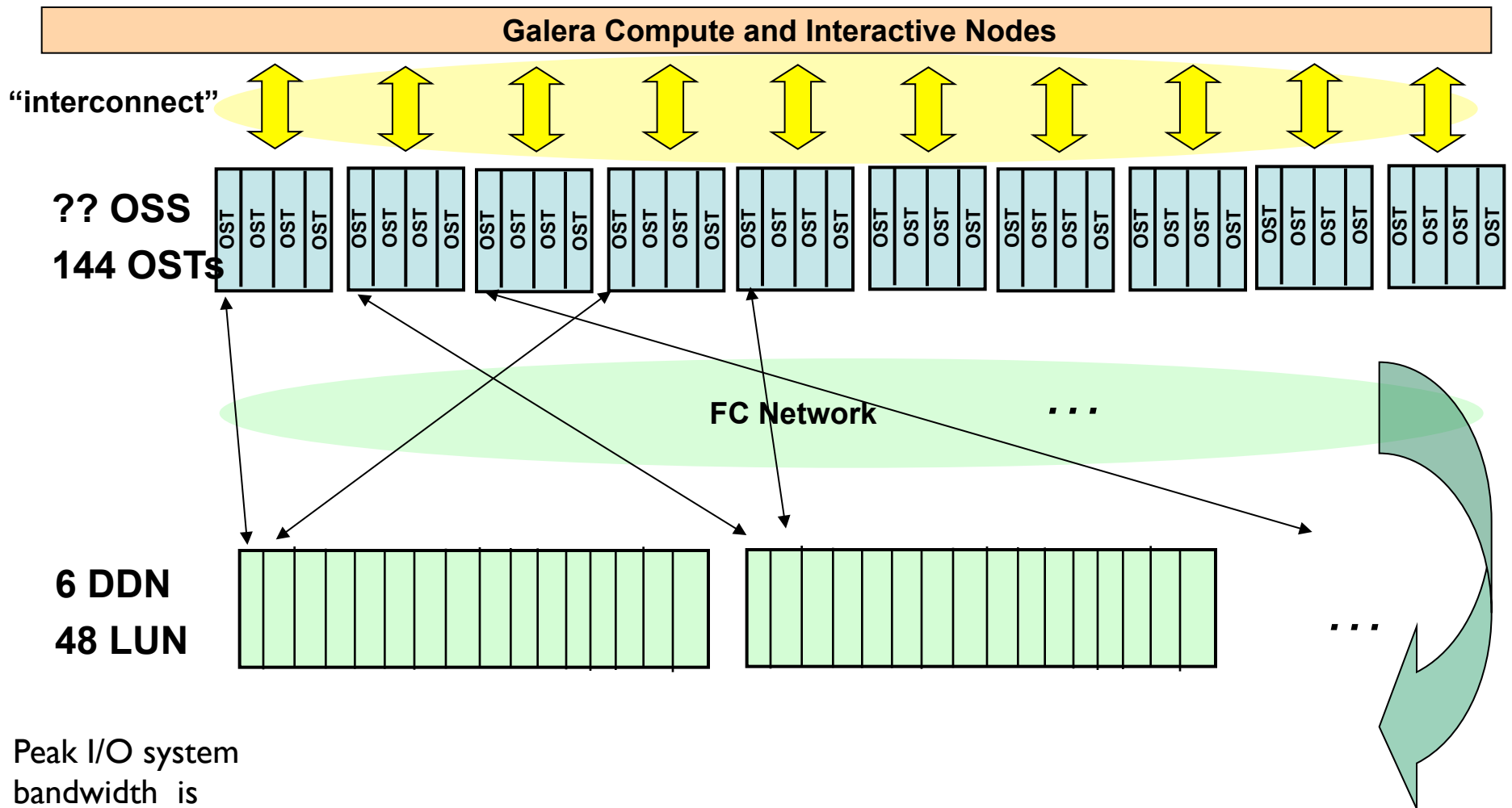
# Generic Parallel File System Architecture







# Galera Cluster Configuration in /work (my best guess)



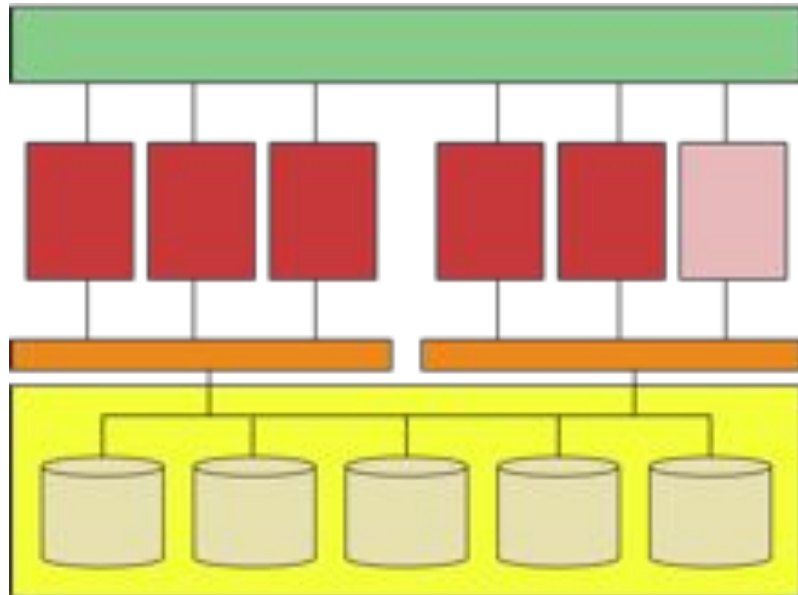
Peak I/O system  
bandwidth is  
~5-7 Gbyte/sec.



# Fault Tolerance and Parallel File Systems

Combination of hardware and software ensures continued operation in face of failures:

- RAID techniques hide disk failures
- Redundant controllers and shared access to storage
- Heartbeat software and quorum directs server failover



**System network** connects storage to compute resources.

**Storage servers** manage independent portions of a shared storage resource. In this diagram, five of six servers are active, while one is passive (a backup).

**Storage controllers** group individual drives into logical units (LUNs) and use RAID techniques to hide drive failures.

**LUNs** are accessible by all storage servers, but only one server accesses any LUN at one time.



# File Buffering and Caching

- **Buffering**
  - Used to improve performance
    - File system collects full blocks of data before transferring data to disk
    - For large writes, can transfer many blocks at once
- **Caching**
  - File system retrieves an entire block of data, even if all data was not requested, data remains in the cache
- Can happen in many places, compute node, I/O server, disk
- Not the same on all platforms
- Important to study your own application's performance rather than look at peak numbers

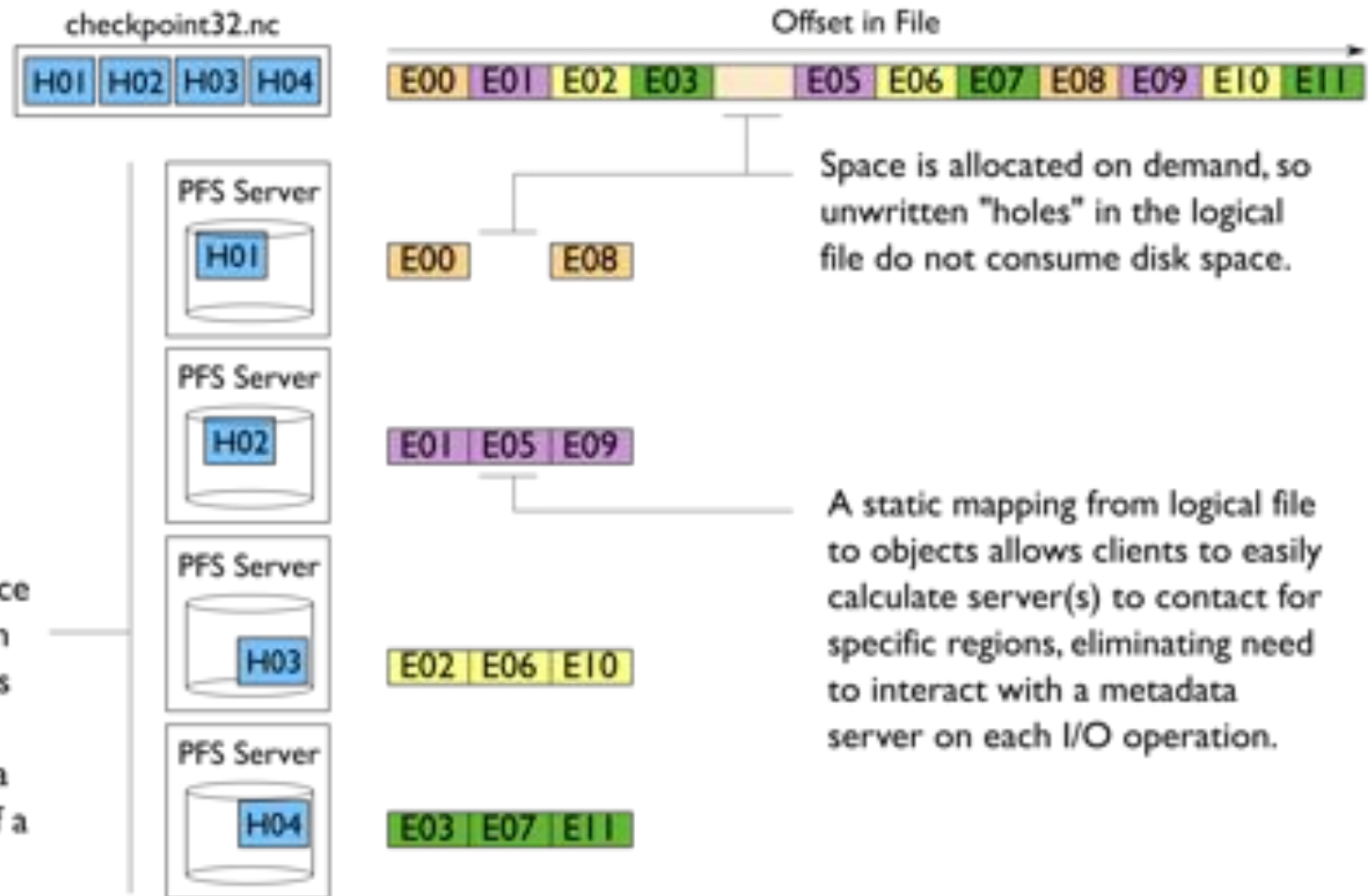


# Data Distribution in Parallel File Systems

Logically a file is an extendable sequence of bytes that can be referenced by offset into the sequence.

Metadata associated with the file specifies a mapping of this sequence of bytes into a set of objects on PFS servers.

Extents in the byte sequence are mapped into objects on PFS servers. This mapping is usually determined at file creation time and is often a round-robin distribution of a fixed extent size over the allocated objects.



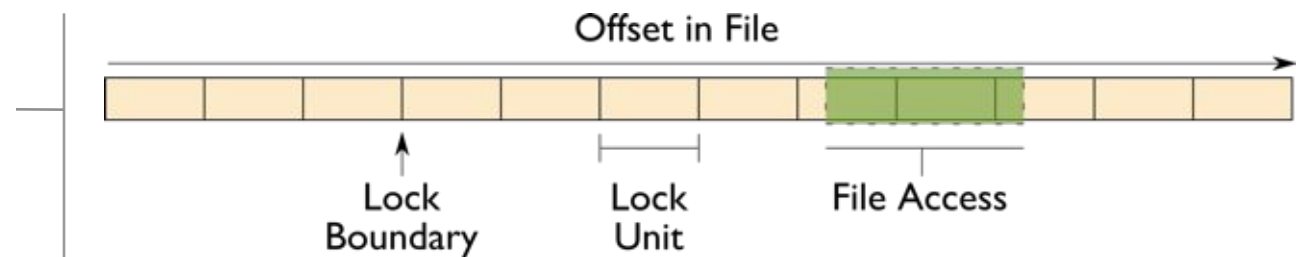


# Locking in Parallel File Systems

**Most parallel file systems use locks to manage concurrent access to files**

- Files are broken up into lock units, (also called blocks)
- Clients obtain locks on units that they will access before I/O occurs
- Enables caching on clients as well (as long as client has a lock, it knows its cached data is valid)
- Locks are reclaimed from clients when others desire access

If an access touches any data in a lock unit, the lock for that region must be obtained before access occurs.

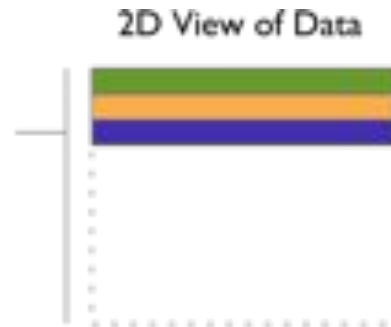






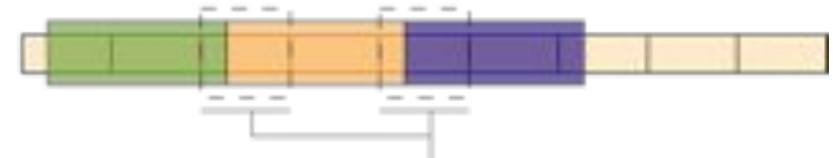
# Locking and Concurrent Access

The left diagram shows a row-block distribution of data for three processes. On the right we see how these accesses map onto locking units in the file.



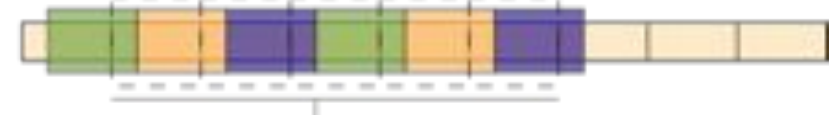
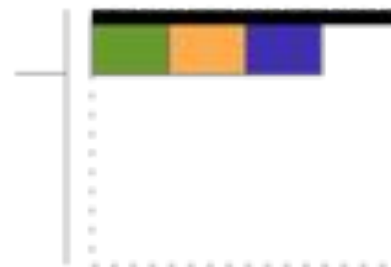
When accesses are to large contiguous regions, and aligned with lock boundaries, locking overhead is minimal.

In this example a header (black) has been prepended to the data. If the header is not aligned with lock boundaries, false sharing will occur.



These two regions exhibit *false sharing*: no bytes are accessed by both processes, but because each block is accessed by more than one process, there is contention for locks.

In this example, processes exhibit a block-block access pattern (e.g. accessing a subarray). This results in many interleaved accesses in the file.



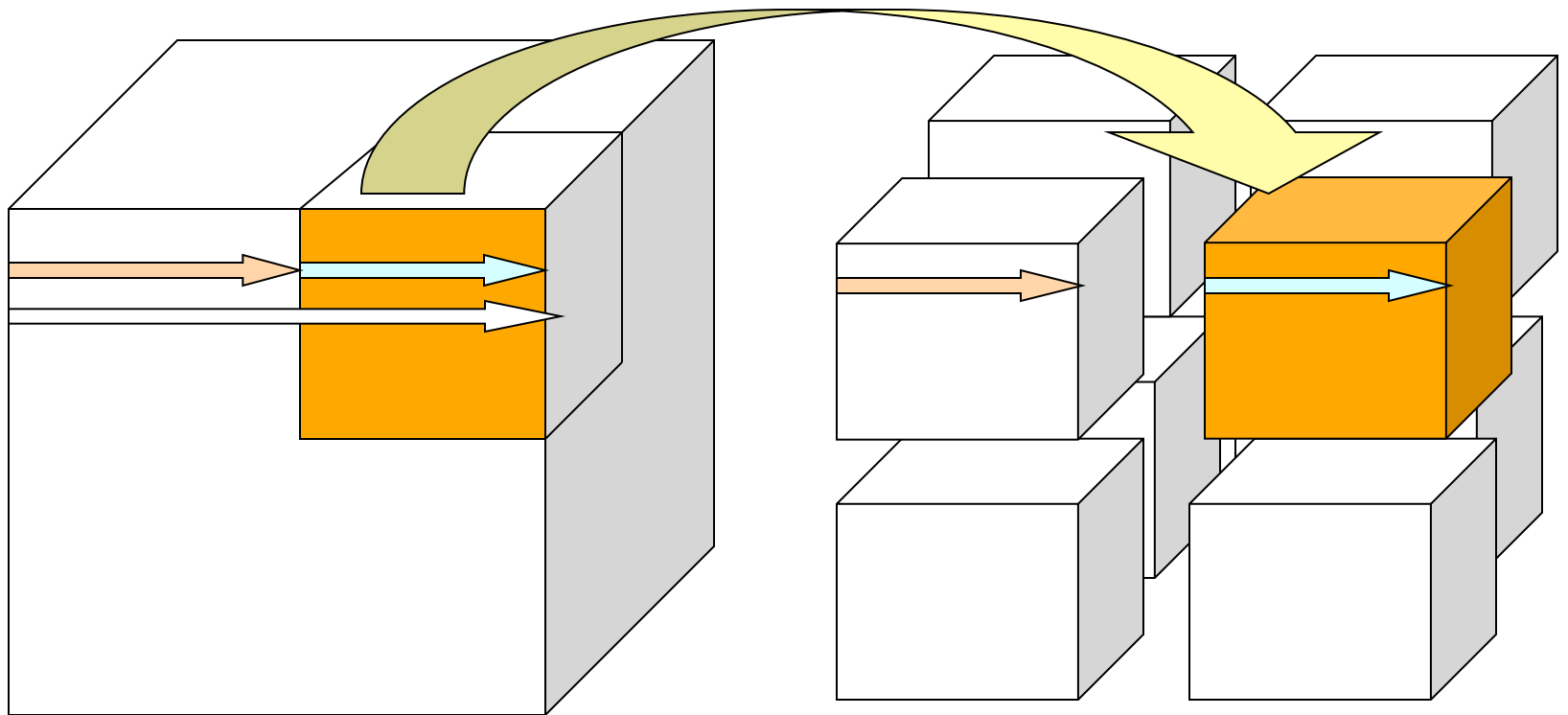
When a block distribution is used, sub-rows cause a higher degree of false sharing, especially if data is not aligned with lock boundaries.

Question -- what would happen if writes were smaller than lock units?



## 3D (reversing the decomp)

Logical



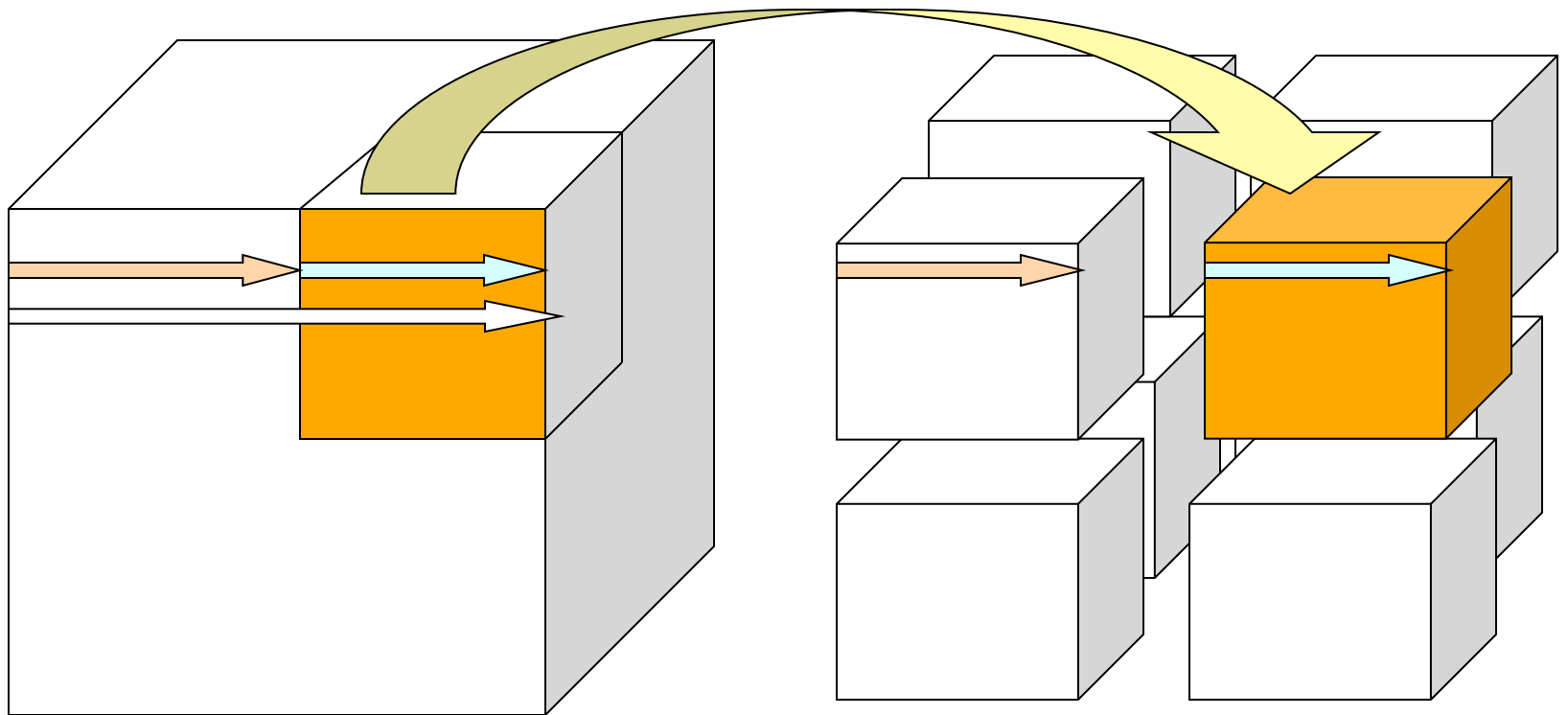
Physical



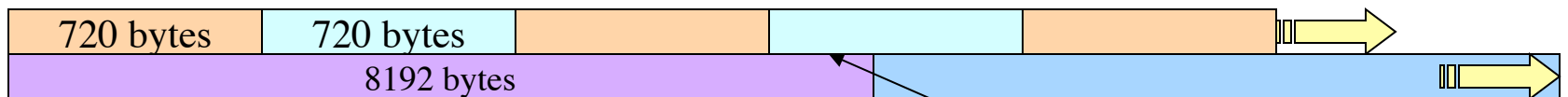


# 3D (block alignment issues)

Logical



Physical



- Block updates require mutual exclusion
- Block thrashing on distributed FS
- I/O efficiency for sparse updates! (8k block required for 720 byte I/O operation)
- Unaligned block accesses can kill performance! (but are necessary in practical I/O solutions)

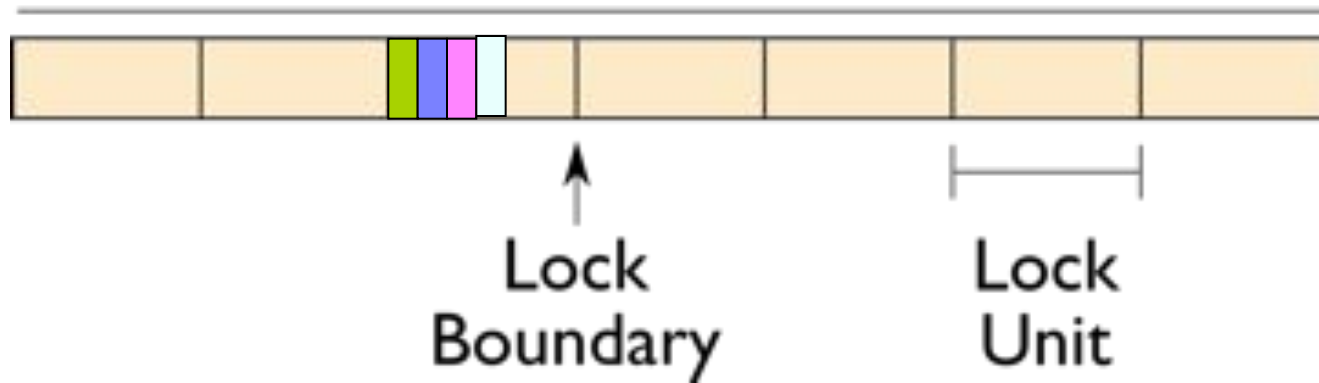
Writes not aligned  
to block boundaries

Slide from John Shalf



# Small Writes

How will the parallel file system perform with small writes (less than the size of a lock unit)?





# Now from the User's point of view



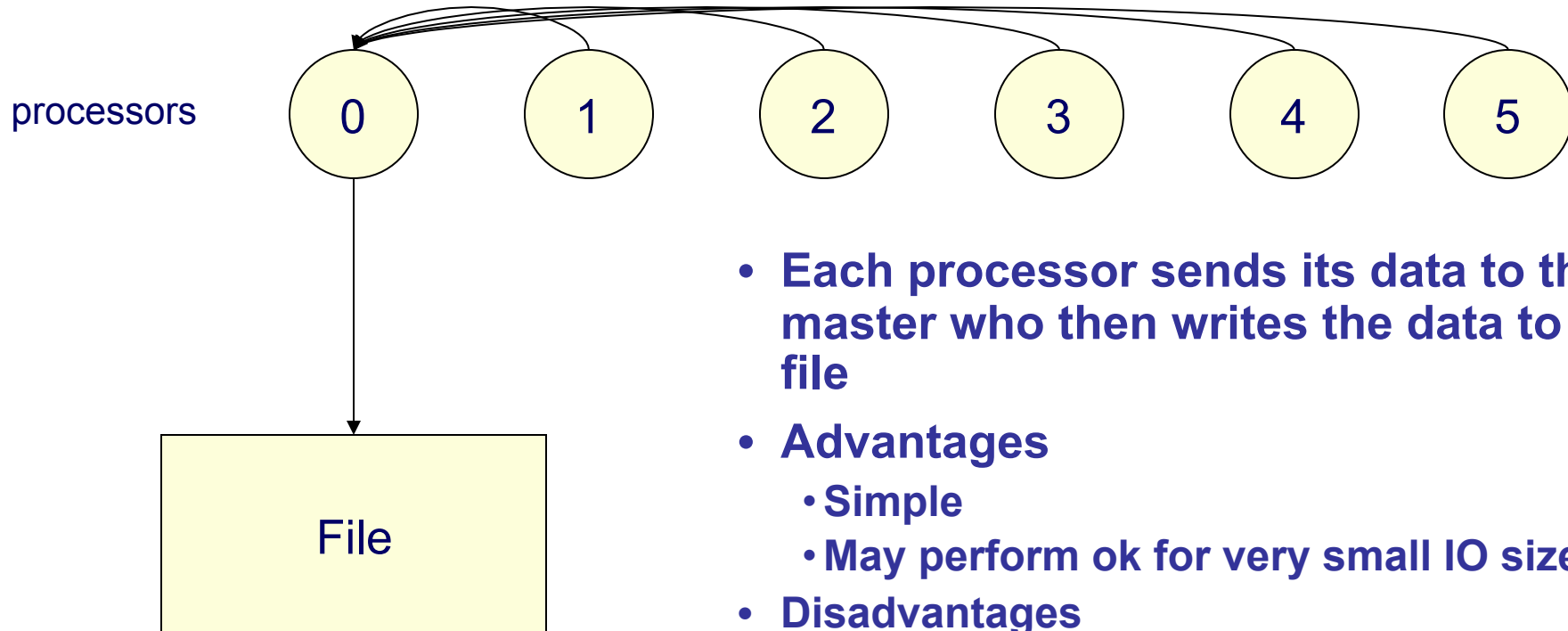


# Why you might need to do I/O

- **Checkpoint/Restart files**
  - System or node could fail; protect your application so you don't have to start from the beginning
  - Need to run longer than wall clock time allows
- **Analysis files**
- **Visualization files**
- **Out-of-core algorithm:**



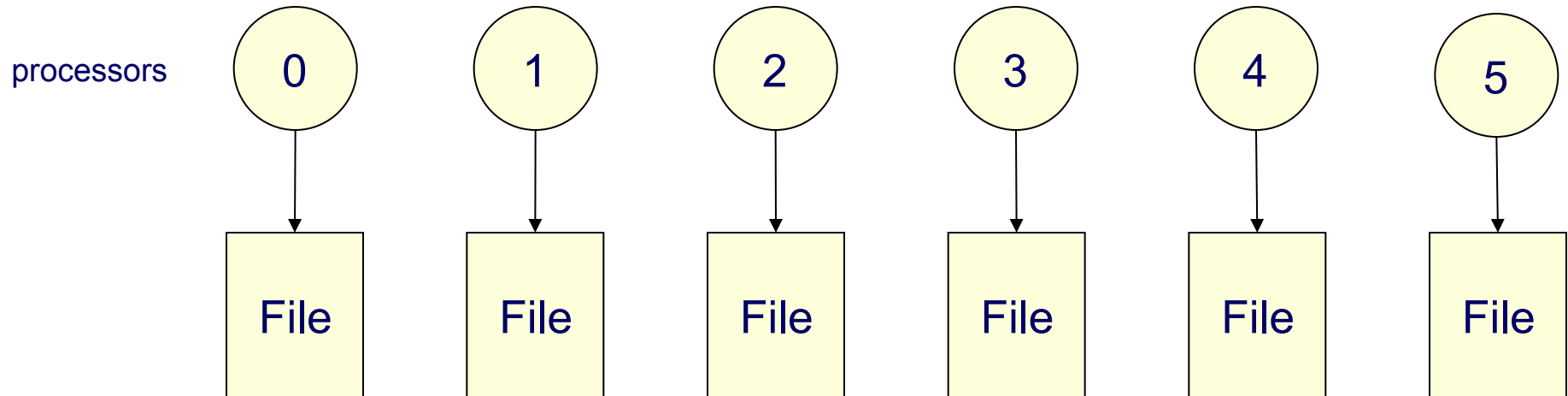
# Serial I/O



- Each processor sends its data to the master who then writes the data to a file
- Advantages
  - Simple
  - May perform ok for very small IO sizes
- Disadvantages
  - Not scalable
  - Not efficient, slow for any large number of processors or data sizes
  - May not be possible if memory constrained



# Parallel I/O Multi-file



- Each processor writes its own data to a separate file
- Advantages
  - Simple to program
  - Can be fast -- (up to a point)
- Disadvantages
  - Can quickly accumulate many files
  - Hard to manage
  - Requires post processing
  - Difficult for storage systems, HPSS, to handle many small files
  - Can overwhelm the file system with many writers

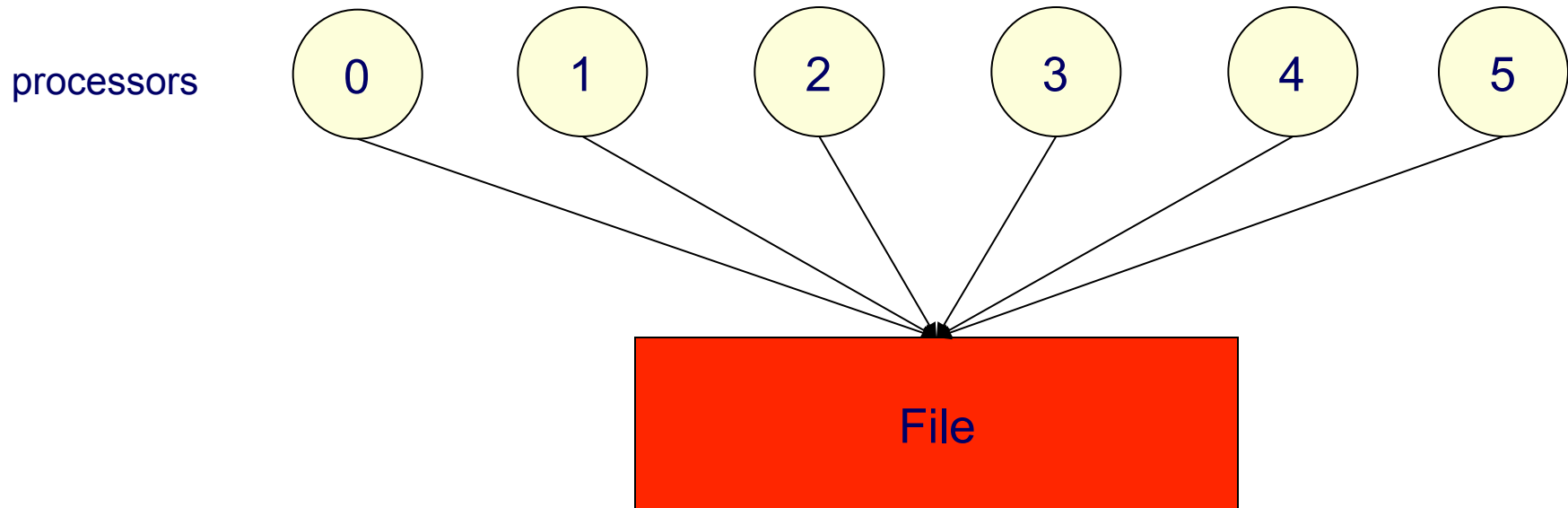


# Flash Center IO Nightmare...

- Large 32,000 processor run on LLNL BG/L
- Parallel IO libraries not yet available
- Intensive I/O application
  - checkpoint files .7 TB, dumped every 4 hours, 200 dumps
    - used for restarting the run
    - full resolution snapshots of entire grid
  - plotfiles - 20GB each, 700 dumps
    - coarsened by a factor of two averaging
    - single precision
    - subset of grid variables
  - particle files 1400 particle files 470MB each
- 154 TB of disk capacity
- 74 million files!
- Unix tool problems
- Took 2 years to sift through data, sew files together



# Parallel I/O Single-file

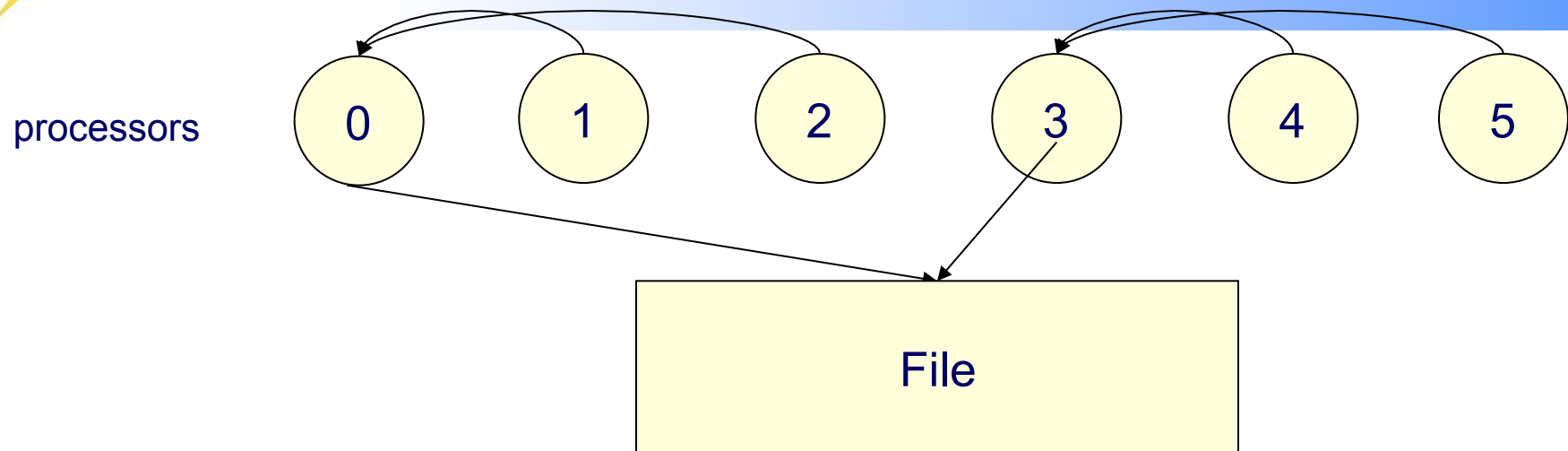


- Each processor writes its own data to the same file using MPI-IO mapping
- Advantages
  - Single file
  - Manageable data
- Disadvantages
  - Shared files may not perform as well as one-file-per-processor models





## Reduced Writers to Single-file



- Best performance when # of writers is multiple of (1-4) # of IO nodes
- Subset of processors writes data to single file
- Advantages
  - Single file; manageable data
  - Better performance than all tasks writing for high concurrency jobs
- Disadvantages
  - This is a pain to program
  - User shouldn't have to do this!

**Users don't need to do this at the application layer**



# Stressing the I/O System

- **Computational science applications exhibit complex I/O patterns that are unique, and how we describe these patterns influences performance.**
- **Accessing from large numbers of processes has the potential to overwhelm the storage system. How we describe the relationship between accesses influences performance.**
- **In some cases we simply need to reduce the number of processes accessing the storage system in order to match number of servers or limit concurrent access.**

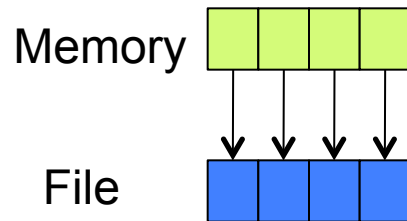


# Access Patterns

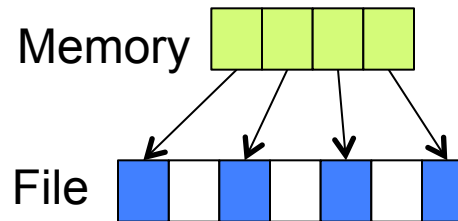


# Access Patterns

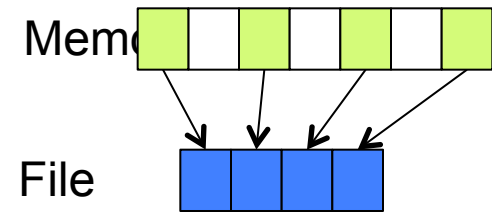
## *Contiguous*



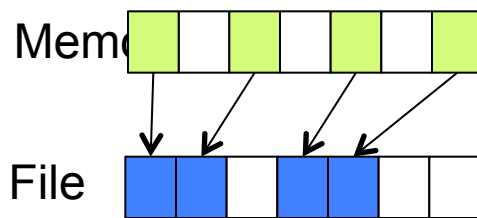
## *Contiguous in memory, not in file*



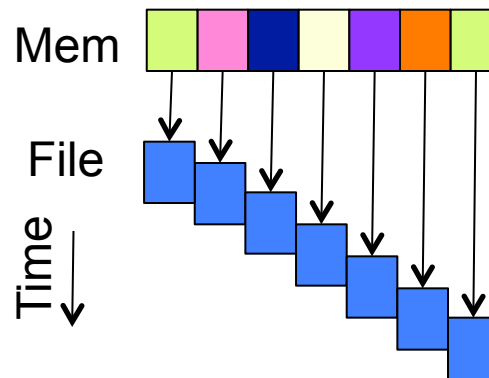
## *Contiguous in file, not in memory*



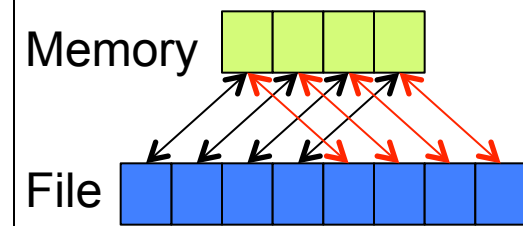
## *Dis-contiguous*



## *Bursty*



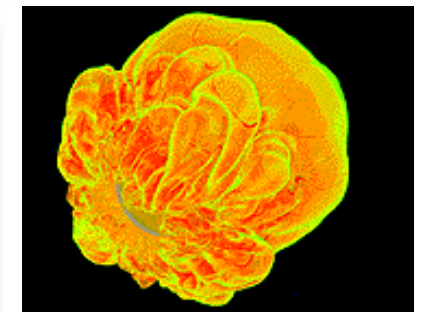
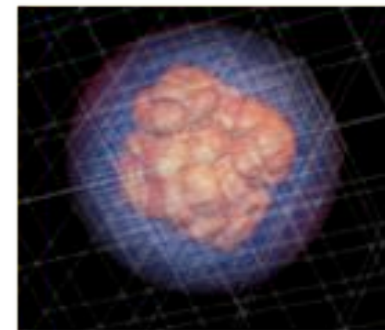
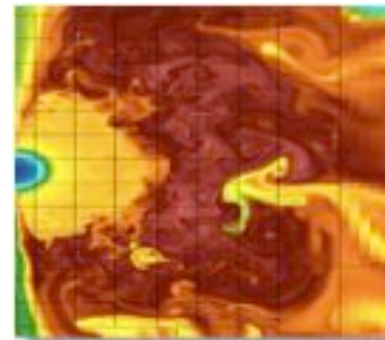
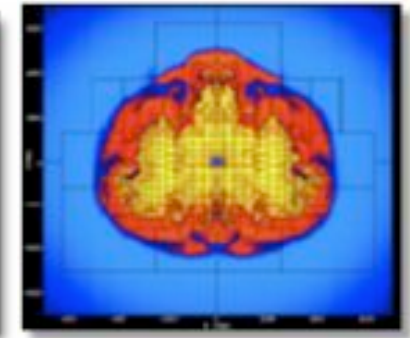
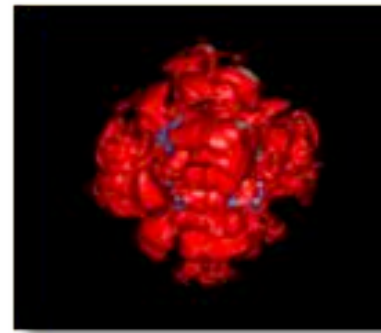
## *Out-of-Core*





# I/O in Astrophysics

- What are the common characteristics of astrophysics applications?
  - Often have LOTS of data
    - Use all memory per core
    - Dump checkpoint and analysis files
  - Usually grid based
    - Structured/unstructure/adaptive grids
    - Can often collect data into large buffers and chunks
    - Regularly ordered, can be contiguous
    - Possible non-contiguous data with 3d decomposition
  - Particles data can be irregular
  - Some applications are out of core





# MPI-IO



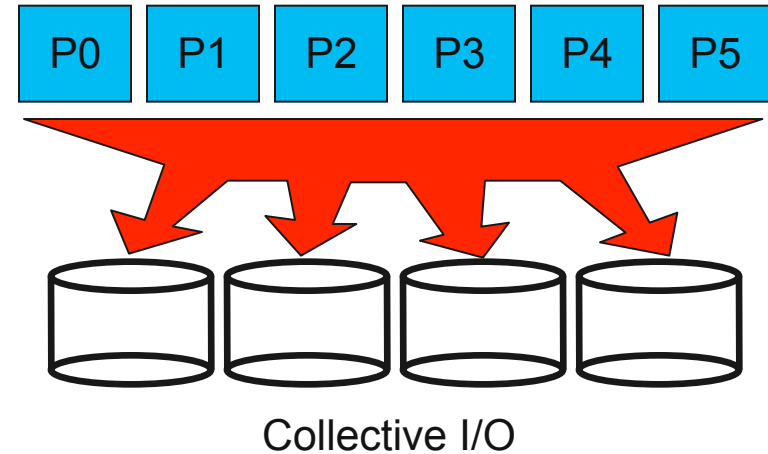
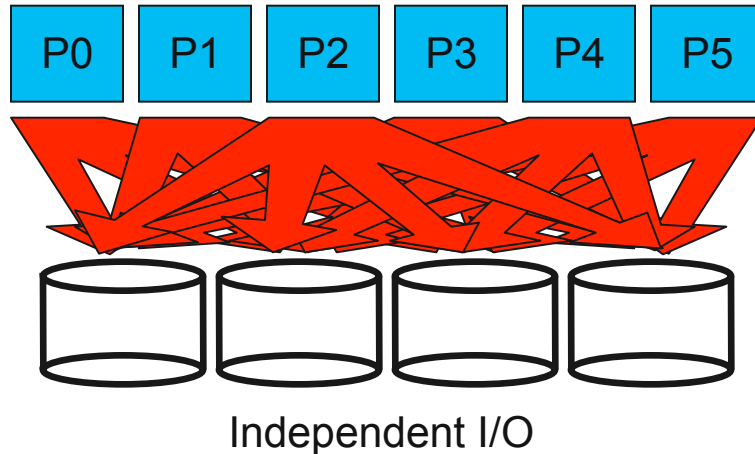
# What is MPI-IO?

- **Parallel I/O interface for MPI programs**
- **Allows users to write shared files with a simple interface**
- **Key concepts:**
  - **MPI communicators**
  - **Derived data types**
  - **File views**
    - **Define which parts of a file are visible to a given processor**
    - **Can be shared by all processors, distinct or partially overlapped**
  - **Collective I/O for optimizations**





# Independent and Collective I/O



- **Independent I/O** operations specify only what a single process will do
  - Independent I/O calls obscure relationships between I/O on other processes
- Many applications have phases of computation and I/O
  - During I/O phases, all processes read/write data
- **Collective I/O** is coordinated access to storage by a group of processes
  - Collective I/O functions are called by all processes participating in I/O
  - **Allows I/O layers to know more about access as a whole, more opportunities for optimization in lower software layers, better performance**



# MPI-IO Optimizations

- **Collective Buffering**
  - Consolidates I/O requests from all procs
  - Only a subset of procs (called aggregators) write to the file
  - Key point is to limit writers so that procs are not competing for same I/O block of data
  - Various algorithms exist for aligning data to block boundaries
  - Collective buffering is controlled by MPI-IO hints: `romio_cb_read`, `romio_cb_write`, `cb_buffer_size`, `cb_nodes`, `cb_config_list`



# When to use collective buffering

- When to use collective buffering
  - The smaller the write, the more likely it is to benefit from collective buffering
  - Large contiguous I/O will not benefit from collective buffering. (If write size is larger than I/O block then there will not be contention from multiple procs for that block.)
  - Non-contiguous writes of any size will not see a benefit from collective buffering





# MPI-IO Summary

- MPI-IO is in the middle of the I/O stack
- Provides optimizations typically low performing I/O patterns (non-contiguous I/O and small block I/O)
- You could use MPI-IO directly, but better to use a high level I/O library



# High Level Parallel I/O Libraries (HDF5)

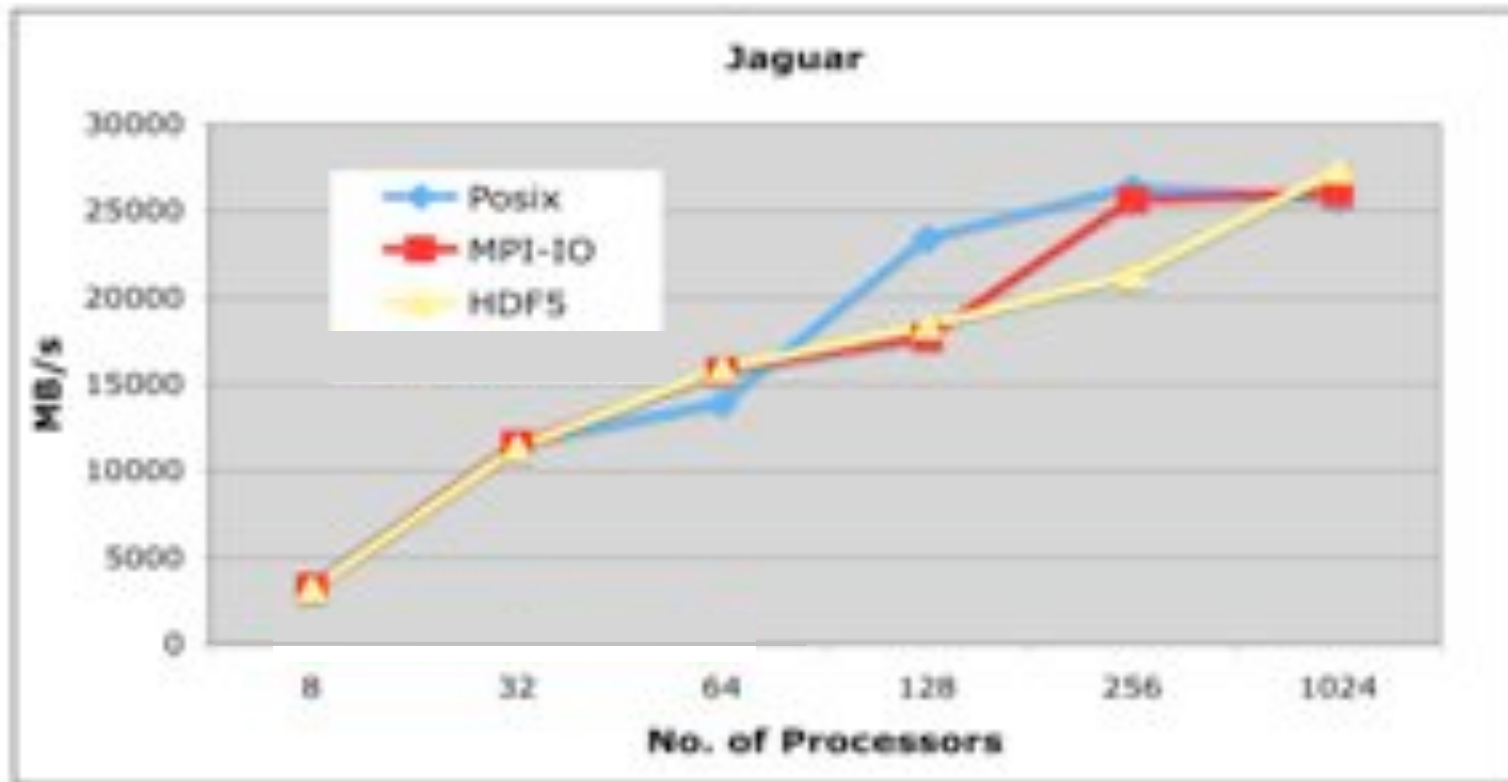


# What is a High Level Parallel I/O Library?

- **An API which helps to express scientific simulation data in a more natural way**
  - Multi-dimensional data, labels and tags, non-contiguous data, typed data
- **Typically sits on top of MPI-IO layer and can use MPI-IO optimizations**
- **Offer**
  - Simplicity for visualization and analysis
  - Portable formats - can run on one machine and take output to another
  - Longevity - output will last and be accessible with library tools and no need to remember version number of code



# IO Library Overhead



*Very little, if any overhead from HDF5 for one file per processor IO compared to Posix and MPI-IO*

Data from Hongzhang Shan





# Common Storage Formats

- **ASCII:**
  - Slow
  - Takes more space!
  - Inaccurate
- **Binary**
  - Non-portable (eg. byte ordering and types sizes)
  - Not future proof
  - Parallel I/O using MPI-IO
- **Self-Describing formats**
  - NetCDF/HDF4, HDF5, Parallel NetCDF
  - Example in HDF5: API implements Object DB model in portable file
  - Parallel I/O using: pHDF5/pNetCDF (hides MPI-IO)
- **Community File Formats**
  - FITS, HDF-EOS, SAF, PDB, Plot3D
  - Modern Implementations built on top of HDF, NetCDF, or other self-describing object-model API

Many NERSC users at this level. We would like to encourage users to transition to a higher IO library







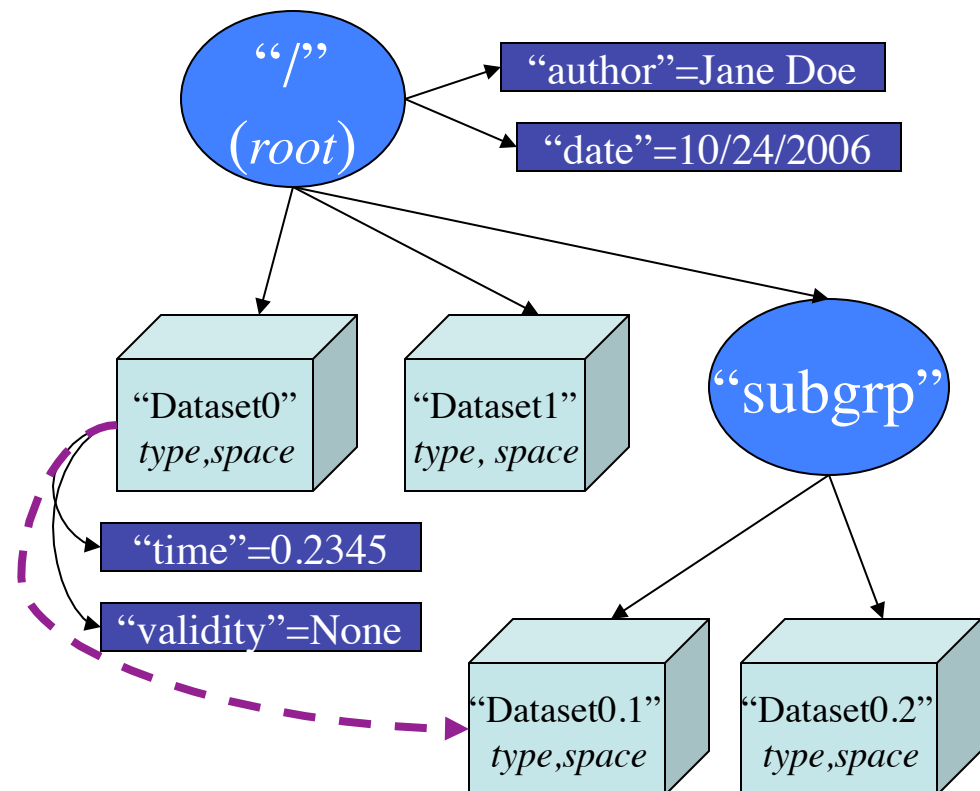
## But what about performance?

- Hand tuned I/O for a particular application and architecture will likely perform better, but ...
- Purpose of I/O libraries is not only portability, longevity, simplicity, but productivity
- Using own binary file format forces user to understand layers below the application to get optimal IO performance
- Every time code is ported to a new machine or underlying file system is changed or upgraded, user is required to make changes to improve IO performance
- Let other people do the work
  - HDF5 can be optimized for given platforms and file systems by library developers
- Goal is for shared file performance to be 'close enough'



# HDF5 Data Model

- **Groups** 
  - Arranged in directory hierarchy
  - root group is always '/'
- **Datasets** 
  - Dataspace
  - Datatype
- **Attributes** 
  - Bind to Group & Dataset
- **References** 
  - Similar to softlinks
  - Can also be subsets of data





# Example HDF5 file output

```
HDF5 "example_file.h5" {  
  GROUP "/" {  
    DATASET "hamiltonian_000" {  
      DATATYPE H5T_IEEE_F64LE  
      DATASPACE SIMPLE { ( 10 ) / ( 10 ) }  
      DATA {  
        (0): 0, 0, 0, 0, 0, 0, 0, 0, 0, 0  
      }  
    }  
    DATASET "hamiltonian_001" {  
      DATATYPE H5T_IEEE_F64LE  
      DATASPACE SIMPLE { ( 10 ) / ( 10 ) }  
      DATA {  
        (0): 1, 1, 1, 1, 1, 1, 1, 1, 1, 1  
      }  
    }  
    DATASET "hamiltonian_002" {  
      DATATYPE H5T_IEEE_F64LE  
      DATASPACE SIMPLE { ( 10 ) / ( 10 ) }  
      DATA {  
        (0): 2, 2, 2, 2, 2, 2, 2, 2, 2, 2  
      }  
    }  
  }  
}
```



## The HDF Group

- HDF5 is maintained by a non-profit company called the *HDF Group*
- Example code and documentation can be found here:
- <http://www.hdfgroup.org/HDF5/>



# Recommendations

- **Think about the big picture**
  - Run time vs Post Processing trade off
  - Decide how much IO overhead you can afford
  - Data Analysis
  - Portability
  - Longevity
    - H5dump works on all platforms
    - Can view an old file with h5dump
    - If you use your own binary format you must keep track of not only your file format version but the version of your file reader as well
  - Storability



# File Striping on Lustre File System



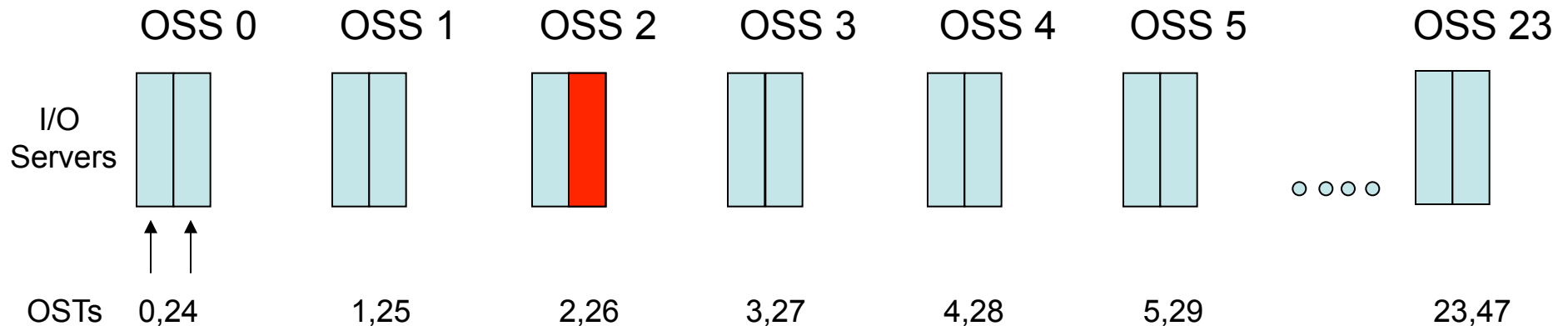


# What is File Striping?

- **Lustre file systems are made up of an underlying set of parallel I/O servers**
  - **OSSs (Object Storage Servers) - nodes dedicated to I/O connected to high speed torus interconnect**
  - **OSTs (Object Storage Targets) software abstraction of physical disk (1 OST maps to 1 LUN)**
- **File is said to be striped when read and write operations access multiple OSTs concurrently**
- **Striping can increase I/O performance since writing or reading from multiple OSTs simultaneously increases the available I/O bandwidth**



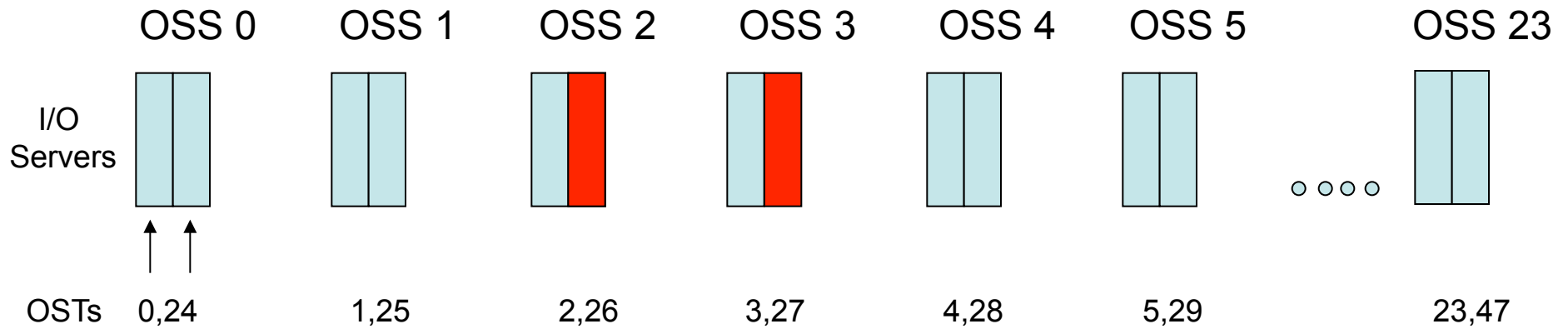
## Default Striping on Galera /work



- **3 parameters characterize striping pattern of a file**
  - **Stripe count**
    - Number of OSTs file is split across
    - Default is 1
  - **Stripe size**
    - Number of bytes to write on each OST before cycling to next OST
    - Default is 1MB
  - **OST offset**
    - Indicates starting OST
    - Default is round robin across all requests on system



## A Stripe Count of 2



- **Pros**

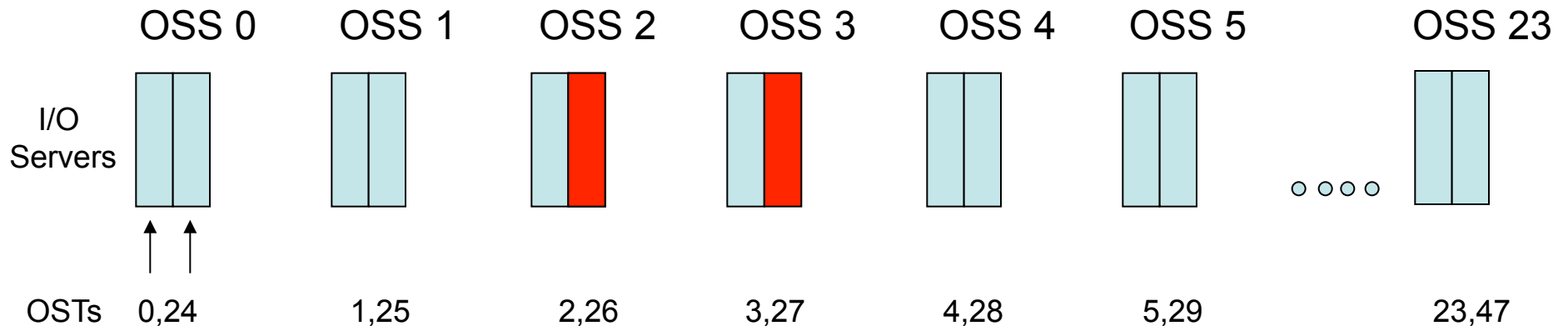
- Get 2 times the bandwidth you could from using 1 OST
- Max bandwidth to 1 OST ~ 350 MB/Sec
- Using 2 OSTs ~700 MB/Sec

- **Cons**

- For better or worse your file now is in 2 different places
- Metadata operations like 'ls -l' on the file could be slower
- For small files (<100MB) no performance gain from striping



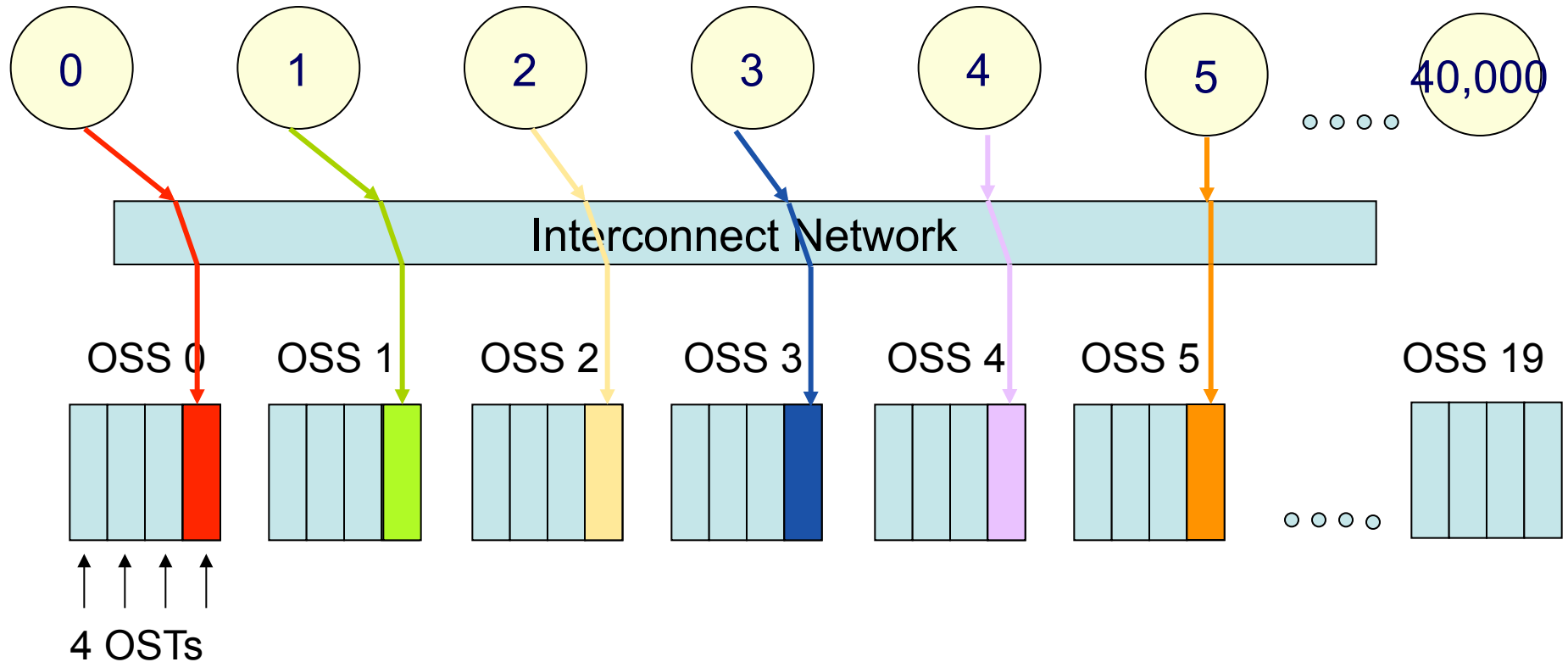
# What's the best stripe count for a file system?



- Depends on the work load of the system and size of disks
- Balance
  - Should work reasonably well for most users
- Protection
  - Each OST is backed up by a physical disk (LUN)
  - Stripe count of 1 leave us vulnerable to single user writing out huge amount of data filling the disk
- Striping of 2 is a reasonable compromise, although not good for large shared files



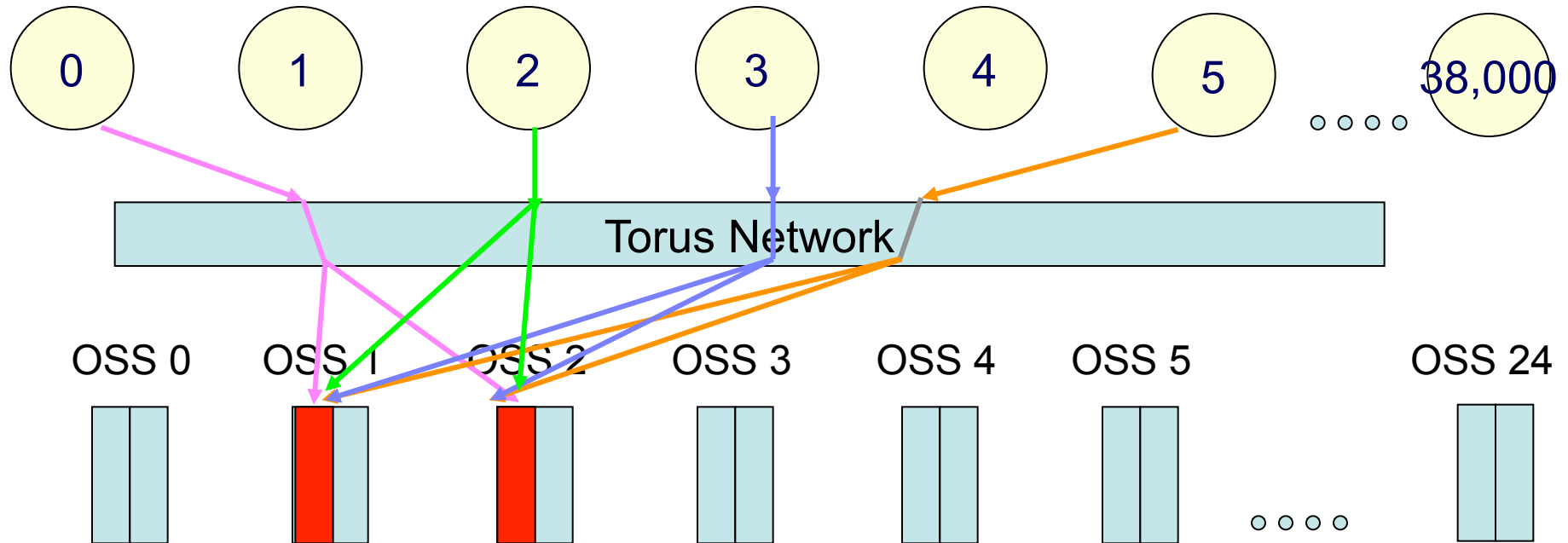
# One File-Per-Processor IO with Stripe Count of 1



- Use all OSTs but don't add more contention than is necessary***



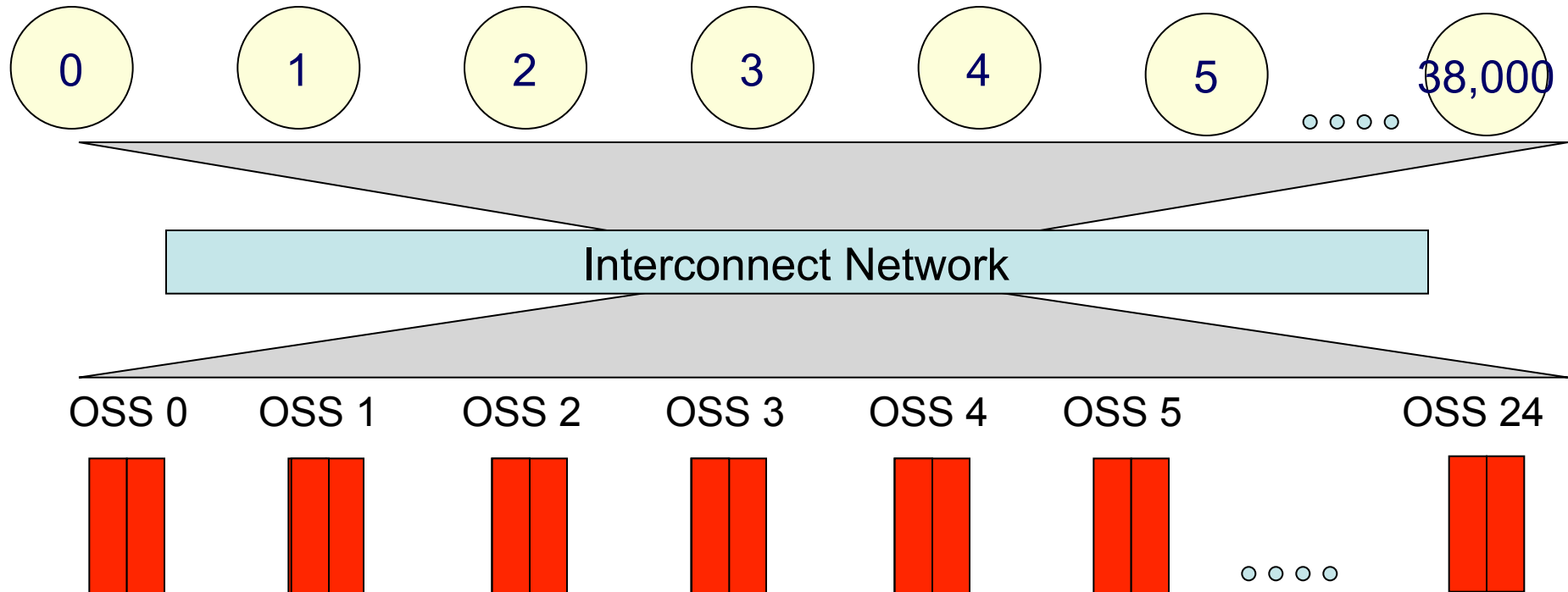
## Shared File I/O with Default Stripe Count 2



- *All processors writing shared file will write to 2 OSTs*
- *No matter how much data the application is writing, it won't get more than ~700 MB/sec (2 OSTs \* 350 MB/Sec)*
- *Less sophisticated than you might think - no optimizations for matching processor writer to same OST*
- *Need to use more OSTs for large shared files*



## Shared File I/O with Stripe Count 48



- *Now Striping over all OSTs*
- *Increased available bandwidth to application*





# Changing the Default Stripe Count

- A number of applications will see benefits from changing the default striping
- Striping can be set at a file or directory level
- When striping set on a directory: all files created in that directory will inherit striping set on the directory

`lfs setstripe <directory|file> -c stripe-count -s stripe-size -o offset`

- **Stripe count - # of OSTs file is split across**
- Stripe size - # bytes written on each OST before cycling to next OST
- OST offset - indicates starting OST

Example: change stripe count to 10

`lfs setstripe mydirectory -c 10`



# Striping Summary

- **Galera Default Striping**
  - Stripe count 1 (data not split over OSTs)
  - Stripe size - 1MB
  - OST offset - round robin starting
- **One File-Per-Processor I/O or shared files < 10 GB**
  - Keep default, stripe count 1
- **Medium shared files: 10GB – 100sGB**
  - Set stripe count ~4-20
- **Large shared files > 1TB**
  - Set stripe count to 20 or higher, maybe all OSTs?
- **You'll have to experiment a little**



# Best Practices

- **Do large I/O: write fewer big chunks of data (1MB+) rather than small bursty I/O**
- **Do parallel I/O.**
  - Serial I/O (single writer) can not take advantage of the system's parallel capabilities.
- **Stripe large files over many OSTs.**
- **If job uses many cores, reduce the number of tasks performing IO**
- **Use a single, shared file instead of 1 file per writer, esp. at high parallel concurrency.**
- **Use an IO library API and write flexible, portable programs.**



# Cover Stories from NERSC Research



NERSC is enabling new high quality science across disciplines, with over *1,600 refereed publications* last year



## **MPI Poll**

**(Steering Friday's Parallelism Lecture)**

- A. ASTROSIM 2010 is the first time I've been exposed to MPI applications**
- B. I use MPI, but don't know many details**
- C. I have used MPI extensively, but would be interested in learning more**
- D. If I have to hear one more MPI talk, I'll go have coffee instead**



# OpenMP

**A. What's OpenMP?**

**B. I've used/tried OpenMP, but don't know many details**

**C. I write hybrid MPI/OpenMP codes with fluency.**



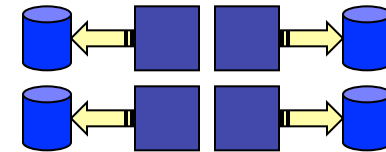
# Extra



# Common Physical Layouts For Parallel I/O

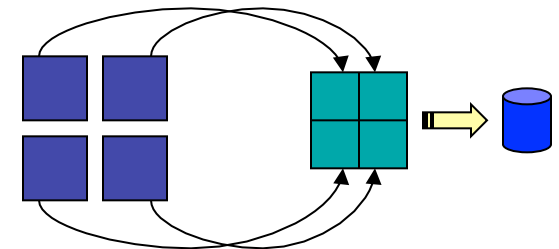
- **One File Per Process**

- Terrible for HPSS!
- Difficult to manage



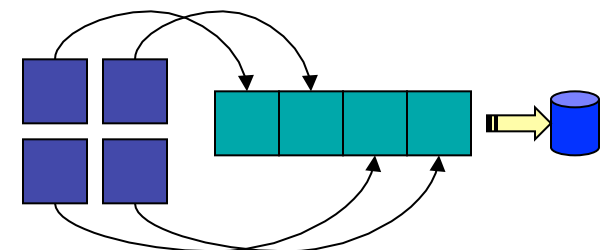
- **Parallel I/O into a single file**

- Raw MPI-IO
- pHDF5 pNetCDF



- **Chunking into a single file**

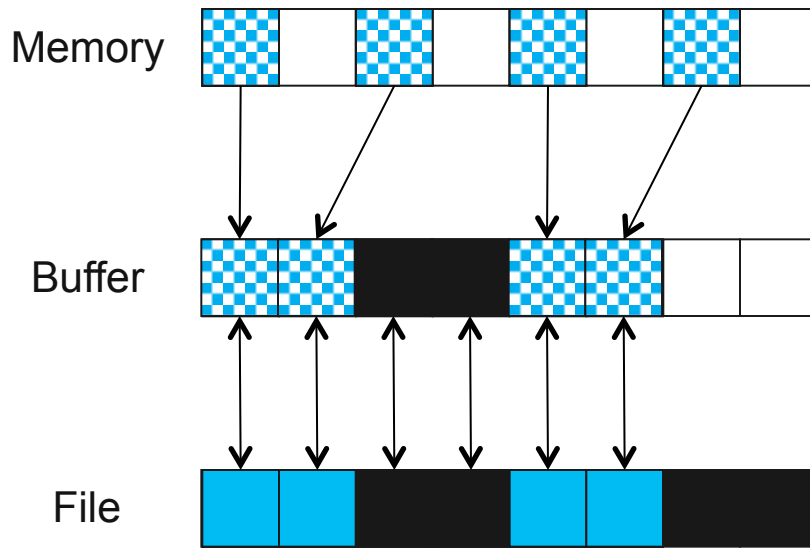
- Saves cost of reorganizing data
- Depend on API to hide physical layout
- (eg. expose user to logically contiguous array even though it is stored physically as domain-decomposed chunks)







# Data Sieving Write Operations

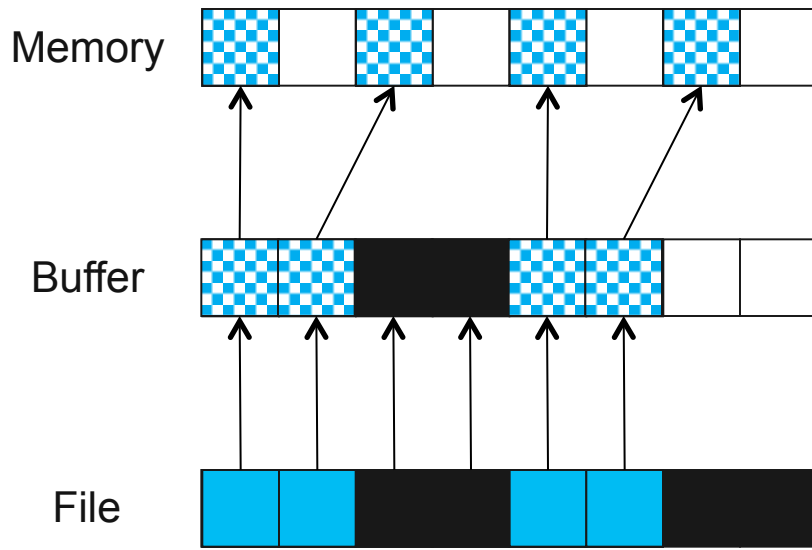


Data Sieving Write Transfers

- **Data sieving for writes is more complicated**
  - Must read the entire region first
  - Then make changes in buffer
  - Then write the block back
- **Requires locking in the file system**
  - Can result in false sharing (interleaved access)



# Noncontiguous I/O Optimization: Data Sieving

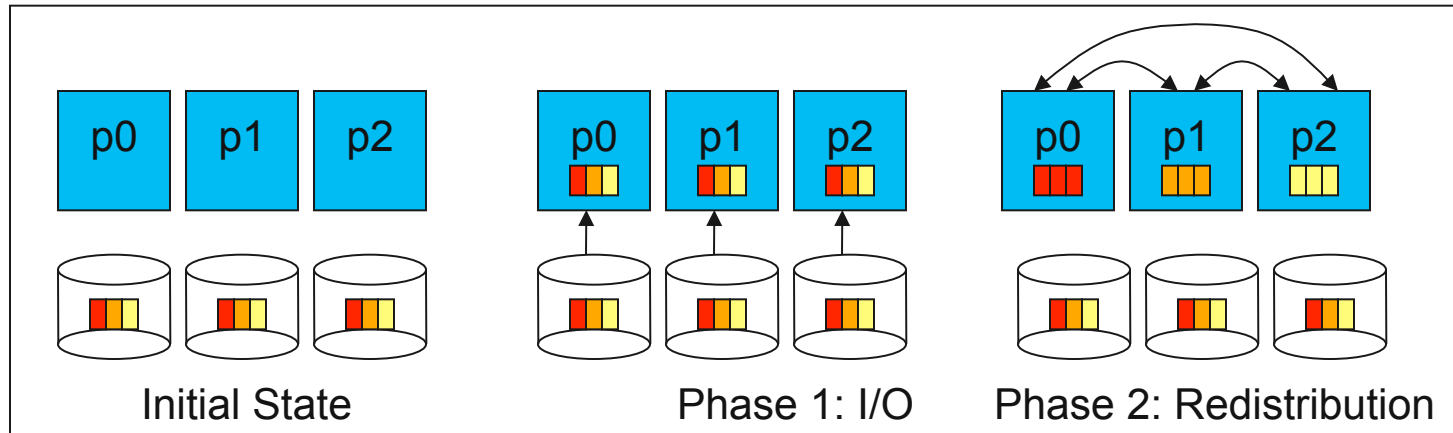


Data Sieving Read Transfers

- **Data sieving is used to combine lots of small accesses into a single larger one**
  - Remote file systems (parallel or not) tend to have high latencies
  - Reducing # of operations important
- **Similar to how a block-based file system interacts with storage**
- **Trade off - read big data chunks, but need more memory**



# Collective I/O Optimization: Two-Phase I/O



Two-Phase Read Algorithm

- **Problems with independent, noncontiguous access**
  - Lots of small accesses
  - Independent data sieving reads lots of extra data, can exhibit false sharing
- **Idea: Reorganize access to match layout on disks**
  - Single processes use data sieving to get data for many
- **Second “phase” redistributes data to final destinations**
- **Two-phase writes operate in reverse (redistribute then I/O)**